

Dischargeable Obligations in Abductive Logic Programming

Marco Alberti^{1(✉)}, Marco Gavanelli², Evelina Lamma², Fabrizio Riguzzi¹,
and Riccardo Zese²

¹ Dipartimento di Matematica e Informatica, University of Ferrara,
Via Saragat 1, 44122 Ferrara, Italy

`{marco.alberti,fabrizio.riguzzi}@unife.it`

² Dipartimento di Ingegneria, University of Ferrara,

Via Saragat 1, 44122 Ferrara, Italy

`{marco.gavanelli,evelina.lamma,riccardo.zese}@unife.it`

Abstract. Abductive Logic Programming (ALP) has been proven very effective for formalizing societies of agents, commitments and norms, in particular by mapping the most common deontic operators (obligation, prohibition, permission) to abductive expectations.

In our previous works, we have shown that ALP is a suitable framework for representing norms. Normative reasoning and query answering were accommodated by the same abductive proof procedure, named SCIFF.

In this work, we introduce a defeasible flavour in this framework, in order to possibly discharge obligations in some scenarios. Abductive expectations can also be qualified as dischargeable, in the new, extended syntax. Both declarative and operational semantics are improved accordingly, and proof of soundness is given under syntax allowedness conditions.

The expressiveness and power of the extended framework, named SCIFF^D, is shown by modeling and reasoning upon a fragment of the Japanese Civil Code. In particular, we consider a case study concerning manifestations of intention and their rescission (Sect. 2 of the Japanese Civil Code).

1 Introduction

A normative system is a set of norms encoded in a formal language, together with mechanisms to reason about, apply, and modify them. Since norms preserve the autonomy of the interacting parties (which ultimately decide whether or not to comply), normative systems are an appropriate tool to regulate interaction in multi-agent systems [15]. Usually, norm definitions build upon notions of obligation, permission and prohibition, in the tradition of Deontic Logic [36].

When formalizing norms, a natural approach is to encode them as implications; semantically, implications naturally represent conditional norms, where the antecedent is read as a property of a state of affairs and the consequent as its deontic consequence, and operationally rule-based systems offer support for reasoning and drawing conclusions from norms and a description of the system they

regulate. Applications of computational logic to formalize norms include logic programming for the British Nationality Act [35], argument-based extended logic programming with defeasible priorities [33], defeasible logic [27].

As mentioned above, normative systems have been applied in multi-agent systems [15]. Among the organizational models and [18, 19] exploit Deontic Logic to specify the society norms and rules. Significant portions of EU research projects were devoted to formalizing norms for multiagent systems; namely, the ALFEBITE project [11] was focused on the formalization of an open society of agents using Deontic Logic, and in the IMPACT project [12, 20] an agent's obligations, permissions and prohibitions were specified by corresponding deontic operators.

The EU IST Project SOCS proposed various Abductive Logic Programming (ALP) languages and proof procedures to specify and implement both individual agents [17] and their interaction [4]; both approaches have later been applied to modeling and reasoning about norms with deontic flavours [7, 34].

ALP has been proved a powerful tool for knowledge representation and reasoning [30], taking advantage of 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. Fung and Kowalski proposed the IFF abductive proof-procedure [23] to deal with forward rules, and with non-ground abducibles. It has been later extended [5], and the resulting proof procedure, named *SCIFF*, can deal with both existentially and universally quantified variables in rule heads and Constraint Logic Programming (CLP) constraints [28]. 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 *SCIFF* abducible predicates, normative systems [6], interaction protocols for multi-agent systems [8], Web services choreographies [3], and Datalog[±] ontologies [24].

In this work, we present *SCIFF*^D, an extension of the *SCIFF* framework, which introduces a defeasible flavour in the norm portion of the framework, as a mechanism for discharging obligations: intuitively, rather than removing an abductive expectation representing obligation with a sort of contraction [10], we mark it as discharged to indicate that the lack of a fulfilling act is not a violation of the norms. Both declarative and operational semantics are extended accordingly, and a proof of soundness is given under syntax allowedness conditions.

Thanks to this extension, we are better able to cope with real-life norms, even in the legal domain.

The paper is organized as follows. In Sect. 2, we first recall the *SCIFF* language, also mentioning its declarative semantics and its underlying proof procedure, and discuss a case study from Sect. 2 of the Japanese Civil Code. Then, in Sect. 3, we introduce the *SCIFF*^D syntax, with a novel abducible for discharging obligations (namely, expectations); we also discuss the formalization of a further article from the Japanese Civil Code. Section 4 extends the declarative and operational semantics accordingly, and presents the proof of soundness for

the extended framework. In Sect. 5 we discuss related work and in Sect. 6 we conclude the paper.

2 SCIFF language and semantics

As a running example, we consider, throughout the paper, article 96 (“Fraud or duress”) of the Japanese civil code (see, for example, [29]). In order to model, and discuss it, we first provide an informal description of the SCIFF language; for formal definitions, we refer the reader to [7].

The SCIFF language. In SCIFF, the agent behaviour is described by means of events (actual behaviour) and expectations (expected behaviour):

- events are atoms of the form $\mathbf{H}(\textit{Content}, \textit{Time})$
- expectations are abducible atoms of the following possible forms, which, while not being modal operators, can be given a deontic reading as shown in [7]: $\mathbf{E}(\textit{Content}, \textit{Time})$: positive expectations, with a deontic reading of obligation; $\mathbf{EN}(\textit{Content}, \textit{Time})$: negative expectations, read as prohibition; $\neg\mathbf{E}(\textit{Content}, \textit{Time})$: negation of positive expectation, or explicit absence of obligation; $\neg\mathbf{EN}(\textit{Content}, \textit{Time})$: negation of negative expectation, or explicit permission.

where *Content* is a logic term that describes the event and *Time* is a variable or a term representing the time of the event. CLP constraints can be imposed over variables; for time variables, they represent time constraints, such as deadlines.

A SCIFF program is a pair $\langle KB, \mathcal{IC} \rangle$, where *KB* is a set of logic programming clauses (used to express domain specific knowledge) which can have expectations, but not events, in their bodies, and \mathcal{IC} is a set of implications called Integrity Constraints, which implicitly define the expected behaviour of the interacting parties. Function symbols and arbitrary nesting of terms are allowed. Each Integrity Constraint (IC) in \mathcal{IC} has the form $\textit{Body} \rightarrow \textit{Head}$, where *Body* is a conjunction of literals defined in *KB*, events and expectations, while *Head* is a disjunction of conjunctions of expectations.

Thanks to their implication structure and the deontic reading of expectations shown in [7], ICs can be read as conditional norms [6].

Case Study. In order to model article 96 (“Fraud or duress” from the Japanese Civil Code), we describe the content of events and expectations by means of the following terms:

- $\textit{intention}(A, B, I, Id_I)$: person *A* utters a manifestation of intention to person *B*, with identifier Id_I for action *I*;
- $\textit{do}(A, B)$: person *A* performs act *B*;
- $\textit{induce}(A, B)$: act *A* induces act *B*;
- $\textit{rescind}(A, B, I, F, Id_I, Id_R)$: person *A* rescinds, with identifier Id_R , his or her intention, uttered to *B* and identified by Id_I , to perform action *I*, due to fraud or duress *F*;

- $know(A, F)$: person A becomes aware of fact F ;
- $assertAgainst(A, B, Id_R)$: person A asserts the legal act identified by Id_R against person B .

Legally relevant acts (*intention*, *rescind*, *assertAgainst*) have identifiers.

The following integrity constraint states that a manifestation of intention should, in general, be followed by the performance of the act.

$$\mathbf{H}(\textit{intention}(A, B, I, Id_I), T_1) \rightarrow \mathbf{E}(\textit{do}(A, I), T_2) \wedge T_2 > T_1 \quad (1)$$

Each of the following integrity constraints models one of the paragraphs of Article 96. Here, *fraudOrDuress*/1 is a predicate, defined in KB , which specifies which actions count as fraud or duress.

1. Manifestation of intention which is induced by any fraud or duress may be rescinded.

$$\begin{aligned} & \mathbf{H}(\textit{intention}(A, B, I, Id_I), T_1) \wedge \mathbf{H}(\textit{do}(B, F), T_3) \wedge \mathbf{H}(\textit{induce}(F, I), T_2) \\ & \wedge \textit{fraudOrDuress}(F) \wedge T_3 < T_2 \wedge T_2 < T_1 \wedge T_1 < T_4 \\ & \rightarrow \neg \mathbf{EN}(\textit{rescind}(A, B, I, F, Id_I, Id_R), T_4) \end{aligned} \quad (2)$$

2. In cases any third party commits any fraud inducing any person to make a manifestation of intention to the other party, such manifestation of intention may be rescinded only if the other party knew such fact.

$$\begin{aligned} & \mathbf{H}(\textit{intention}(A, B, I, Id_I), T_1) \wedge \mathbf{H}(\textit{do}(C, F), T_3) \wedge C \neq B \\ & \wedge \mathbf{H}(\textit{know}(B, F), T_5) \wedge \mathbf{H}(\textit{induce}(F, I), T_2) \\ & \wedge \textit{fraudOrDuress}(F) \wedge T_2 < T_1 \wedge T_3 \leq T_5 \wedge T_5 < T_1 \\ & \rightarrow \neg \mathbf{EN}(\textit{rescind}(A, B, I, F, Id_I, Id_R), T_4) \wedge T_4 > T_1 \end{aligned} \quad (3)$$

3. The rescission of the manifestation of intention induced by the fraud pursuant to the provision of the preceding two paragraphs may not be asserted against a third party without knowledge.

$$\begin{aligned} & \mathbf{H}(\textit{rescind}(A, B, I, F, Id_I, Id_R), T_1) \wedge \mathbf{not} \mathbf{H}(\textit{know}(C, F), T_2) \\ & \rightarrow \mathbf{EN}(\textit{assertAgainst}(A, C, Id_R), T_3) \wedge T_1 < T_3 \end{aligned} \quad (4)$$

Declarative Semantics. The abductive semantics of the \mathcal{SCIFF} language defines, given a set \mathbf{HAP} of \mathbf{H} atoms called history and representing the actual behaviour, an abductive answer, i.e., a ground set \mathbf{EXP} of expectations that

- together with the history and KB , entails \mathcal{IC} :

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC} \quad (5)$$

where \models is entailment according to the in 3-valued completion semantics.

- is consistent with respect to explicit negation;

$$\begin{aligned} & \{\mathbf{E}(\text{Content}, \text{Time}), \neg \mathbf{E}(\text{Content}, \text{Time})\} \not\subseteq \mathbf{EXP} \wedge \\ & \wedge \{\mathbf{EN}(\text{Content}, \text{Time}), \neg \mathbf{EN}(\text{Content}, \text{Time})\} \not\subseteq \mathbf{EXP} \end{aligned} \quad (6)$$

- is consistent with respect to the meaning of expectations

$$\{\mathbf{E}(\text{Content}, \text{Time}), \mathbf{EN}(\text{Content}, \text{Time})\} \not\subseteq \mathbf{EXP} \quad (7)$$

- is fulfilled by the history, i.e.

$$\text{if } \mathbf{E}(\text{Content}, \text{Time}) \in \mathbf{EXP} \text{ then } \mathbf{H}(\text{Content}, \text{Time}) \in \mathbf{HAP} \text{ and} \quad (8)$$

$$\text{if } \mathbf{EN}(\text{Content}, \text{Time}) \in \mathbf{EXP} \text{ then } \mathbf{H}(\text{Content}, \text{Time}) \notin \mathbf{HAP} \quad (9)$$

Operational Semantics. Operationally, the *SCIFF* abductive proof procedure finds an abductive answer if one exists, or detects that no one exists (see [5] for soundness and completeness statements), meaning that the history violates the *SCIFF* program. We call the two cases success and failure, respectively. The *SCIFF* proof-procedure is defined through a set of transitions, each rewriting one node of a proof tree into one or more nodes. The basic transitions of *SCIFF* are inherited from the *IFF* [23], and they account for the core of abductive reasoning. Other transitions deal with CLP constraints, and are inherited from the CLP [28] transitions. Due to lack of space, we cannot describe in detail all transitions; we sketch those dealing with the concept of expectation, that are most relevant for the rest of the paper.

In order to deal with the concept of expectation, in each node of the proof tree, the set of abduced expectations \mathbf{EXP} is partitioned into two sets: the fulfilled (**FULF**), and pending (**PEND**) expectations.

Transition *Fulfillment* **E** deals with the fulfillment of **E** expectations: if an expectation $\mathbf{E}(E, T_E) \in \mathbf{PEND}$ and the event $\mathbf{H}(H, T_H)$ is in the current history **HAP**, two nodes are generated: one in which $E = H$, $T_E = T_H$ and $\mathbf{E}(E, T_E)$ is moved to **FULF**, the other in which $E \neq H$ or $T_E \neq T_H$ (where \neq stands for the constraint of disunification).

Transition *Violation* **EN** deals with the violation of **EN** expectations: if an expectation $\mathbf{EN}(E, T_E) \in \mathbf{PEND}$ and the event $\mathbf{H}(H, T_H) \in \mathbf{HAP}$, one node is generated where the constraint $E \neq H \vee T_E \neq T_H$ is imposed, possibly leading to failure.

When there are no more relevant events, *history closure* is applied; in this case, all remaining **E** expectations in **PEND** are considered as violated and failure occurs.

As regards complexity, the *SCIFF* language is an extension of Prolog, and, as such, it is Turing-complete; so a *SCIFF* evaluation, in general, may not terminate. Even in the propositional case, Gottlob and Eiter [21] proved that the complexity of abduction is Σ_2^P -complete.

3 $\text{SCIFF}^{\mathcal{D}}$ Language

In legal reasoning, expectations can be discharged not only because they become fulfilled by matching the actual behaviour of the agent, but also for other reasons. For example, in case a contract is declared null, the agents are no longer expected to perform the actions required in the contract.

We introduce an extension of the SCIFF language to deal with expectations that do not hold any longer. We introduce two new abducible atoms, $\mathbf{D}(\mathbf{E})$ and $\mathbf{D}(\mathbf{EN})$, which mean that an expectation is discharged; for example, the atom

$$\mathbf{D}(\mathbf{E}(X, T))$$

means that the expectation $\mathbf{E}(X, T)$ is no longer required to be fulfilled, as it has been discharged.

The integrity constraints can have \mathbf{D} atoms, which can be abduced.

Example 1. The user might write an IC saying that, if a contract with identifier Id_C is null (represented in this example with an abducible \mathbf{NULL} , carrying the identifier of the contract and that of the reason for nullification), all expectations requiring an action in the context of that contract are discharged:

$$\begin{aligned} \mathbf{NULL}(Id_C, Id_{null}) \wedge \mathbf{E}(do(\text{Agent}, \text{Action}, Id_C), T_{do}) \\ \rightarrow \mathbf{D}(\mathbf{E}(do(\text{Agent}, \text{Action}, Id_C), T_{do})). \end{aligned} \quad (10)$$

We can express that a contract that is explicitly permitted to be rescinded can be nullified as

$$\begin{aligned} \neg \mathbf{EN}(\text{rescind}(A, I, F, Id_I, Id_R), T_r) \wedge \mathbf{H}(\text{rescind}(A, I, F, Id_I, Id_R), T_r) \\ \rightarrow \mathbf{NULL}(Id_I, Id_R) \end{aligned} \quad (11)$$

The combined effect of ICs (10) and (11) is that rescission is only effective when the circumstances grant an explicit permission.

Example 2. As a second case study from the Japanese Civil Code, we consider Article 130, which states “In cases any party who will suffer any detriment as a result of the fulfillment of a condition intentionally prevents the fulfillment of such condition, the counterparty may deem that such condition has been fulfilled”. Article 130 can be modeled as follows:

$$\begin{aligned} \mathbf{H}(do(\text{Agent}_1, \text{Action}_1), T_1) \wedge \mathbf{E}(do(\text{Agent}_2, \text{Action}_2), T_2) \\ \wedge \text{detrimental}(\text{Action}_2, \text{Agent}_1) \wedge \text{prevent}(\text{Action}_1, \text{Action}_2) \\ \rightarrow \mathbf{D}(\mathbf{E}(do(\text{Agent}_2, \text{Action}_2), T_2)) \end{aligned} \quad (12)$$

where $\text{detrimental}/2$ and $\text{prevent}/2$ are predicates defined in the KB to specify when, respectively, an action is detrimental to an agent and when an action prevents another.

Syntactic Restrictions. The following syntactic restriction is used in the proof of soundness.

Definition 1. *Weak \mathbf{D} -allowedness.* An IC containing a \mathbf{D} atom is weakly \mathbf{D} -allowed if there is only one \mathbf{D} atom, it occurs in the head, and the head contains only that atom.

A KB is weakly \mathbf{D} -allowed if none of its clauses contains \mathbf{D} atoms.

A $\text{SCIFF}^{\mathcal{D}}$ program $\langle KB, \mathcal{IC} \rangle$ is weakly \mathbf{D} -allowed if KB is weakly \mathbf{D} -allowed and all the ICs in \mathcal{IC} are weakly \mathbf{D} -allowed.

The following restriction is not necessary for the soundness results proved in Sect. 4.3, but it allows a more efficient treatment of the \mathbf{D} atoms. Note that all the examples presented in this paper satisfy the restriction.

Definition 2. *Strong \mathbf{D} -allowedness.* An IC containing a \mathbf{D} atom is strongly \mathbf{D} -allowed if it is weakly \mathbf{D} -allowed and the (only) expectation in the \mathbf{D} atom occurs identically in the body of the IC.

Intuitively, the given notion of strong \mathbf{D} -allowedness allows one to define ICs that select one expectation and make it discharged, subject to conditions occurring in the body. This syntactic restriction is aimed at capturing the most common scenarios while trying to maintain an efficient execution.

In fact, if the strong allowedness condition is lifted, it is not required for an atom $\mathbf{E}(X)$ to have been abducted before declaring it discharged. If two atoms $\mathbf{E}(X)$ and $\mathbf{D}(\mathbf{E}(Y))$ are abducted, two options have to be explored, as alternatives: either X unifies with Y , and the expectation $\mathbf{E}(X)$ becomes discharged, or X and Y do not unify (e.g., by imposing a dis-unification constraint $X \neq Y$). These cases, the SCIFF proof-procedure opens a choice point; this means that, in case $|E|$ expectations \mathbf{E} are abducted and $|D|$ \mathbf{D} atoms are abducted, $|E||D|$ choice points will be created, each opening 2 alternative branches, which would generate $2^{|E||D|}$ branches (of course, the same could be said for \mathbf{EN} expectations).

We performed an experiment to verify this worst-case analysis. We generated a number of $\mathbf{E}(X)$ and $\mathbf{D}(\mathbf{E}(Y))$ atoms, and measured the time $\text{SCIFF}^{\mathcal{D}}$ took to find the first solution and all solutions (all experiments were run on a Intel Core i7-3720QM CPU @ 2.60 GHz running SWI-Prolog version 7.4.0-rc1 on Linux Mint 18.1 Serena 64 bits). For all solutions (Fig. 1 right), the running time follows closely the foreseen $2^{|E||D|}$, while for one solution (Fig. 1 left), the running time seems dependent mainly on the number of raised expectations and almost independent from the number of \mathbf{D} atoms. Note also the different scales: finding one solution takes at most 3 s with 100 expectations and discharge atoms, while finding all solutions takes almost 3 h with $|E| = |D| = 8$.

From a language viewpoint, the strong allowedness condition restricts the set of expectations that can be discharged to those that have been raised. Without such restriction, one could abduce a generic atom $\mathbf{D}(\mathbf{E}(X))$ saying that one expectation is discharged. Semantically, this would mean that one of the expectations might be discharged, although it is not said which one.

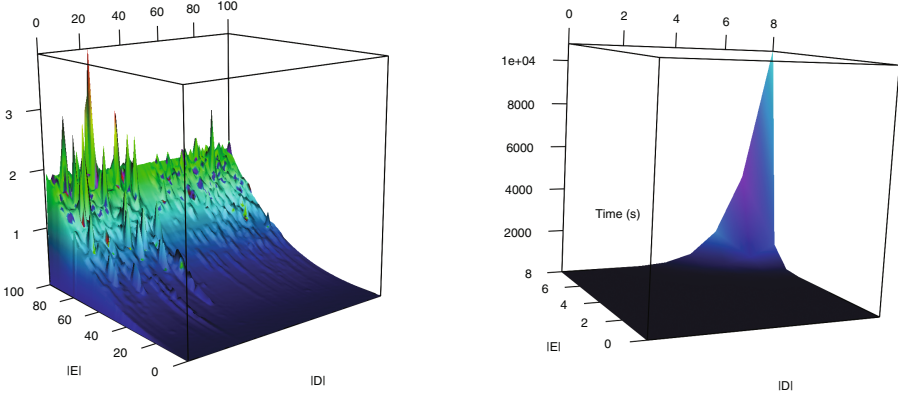


Fig. 1. Experiments with different numbers $|D|$ of **D** atoms and $|E|$ of **E** atoms; time in seconds for finding one solution (left) or all solutions (right).

Dischargetment Scenarios. The remainder of this section is devoted to discuss an example concerning manifestation of intention (Sect. 2 of the Japanese Civil Code), modeled in SCIFF^D and one concerning prevention of fulfillment of conditions. Sections presenting the declarative and operational semantics of the extended framework and proof of soundness then follow.

Example 3. (Example 1 continued). Let us consider the case study of Sect. 2 again, and assume that work on an acquired good G is to be paid by the good's owner O , unless O rescinds the good's purchase and asserts the rescission against the performer of the work M (which, due to integrity constraint (4), is only allowed if the performer was aware of the rescission's cause, such as a fraud). We can express this norm by means of the following integrity constraint:

$$\begin{aligned}
 & \mathbf{H}(\text{work}(M, G, O, W), T_1) \\
 & \rightarrow \mathbf{E}(\text{pay}(O, M, W), T_2) \wedge T_1 < T_2 \\
 & \vee (\mathbf{E}(\text{rescind}(O, B, \text{buy}(G), F, Id_I, Id_R), T_3) \\
 & \quad \wedge \mathbf{E}(\text{assertAgainst}(O, M, Id_R), T_4) \\
 & \quad \wedge T_1 < T_3 \wedge T_3 < T_4)
 \end{aligned} \tag{13}$$

where the term $\text{work}(M, G, O, W)$ represents mechanic M doing work W on the good G owned by O , and the term $\text{pay}(O, M, W)$ represents owner O paying mechanic M for work W .

The KB states that fixing a car's mileage constitutes fraud or duress.

$$\text{fraudOrDuress}(\text{fixMileage}(C)) \tag{14}$$

In the scenario defined by the following history

$$\begin{aligned}
 &\mathbf{H}(\text{do}(\text{bob}, \text{fixMileage}(\text{car})), 1) \\
 &\mathbf{H}(\text{induce}(\text{fixMileage}(\text{car}), \text{buy}(\text{car})), 2) \\
 &\mathbf{H}(\text{intention}(\text{alice}, \text{bob}, \text{buy}(\text{car}), i1), 3) \\
 &\mathbf{H}(\text{work}(\text{mechanic}, \text{car}, \text{alice}, \text{lpgSystem}), 4) \\
 &\mathbf{H}(\text{rescind}(\text{alice}, \text{bob}, \text{buy}(\text{car}), \text{fixMileage}(\text{car}), i1, i2), 5)
 \end{aligned} \tag{15}$$

Alice's rescission is explicitly permitted by IC (2), because her manifestation of intention was induced by Bob's fraudulent act of fixing the car's mileage; the expectation $\mathbf{E}(\text{do}(\text{alice}, \text{buy}(\text{car})), T)$, raised because of IC (1), is discharged because of ICs (10) and (11).

However, since the mechanic was not aware of Bob's fraud, IC (4) prevents Alice from asserting the rescission against him, so the second disjunct in IC (13) cannot hold, and Alice still has to pay him for installing the LPG system ($\mathbf{E}(\text{pay}(\text{alice}, \text{mechanic}, \text{lpgSystem}, T_2))$). For the history that contains all the events in formula (15), plus $\mathbf{H}(\text{know}(\text{mechanic}, \text{fixMileage}(\text{car})), 1)$ (i.e., the mechanic is now aware of the car's mileage being fixed), alice is not prohibited from asserting the rescission against the mechanic by integrity constraint (4). With the event $\mathbf{H}(\text{assertAgainst}(\text{alice}, \text{mechanic}, i2), 6)$, the second disjunct in the head of integrity constraint (13), is satisfied and alice is not obliged to pay the mechanic for his work.

Example 4. Consider the following scenario, where an order by a customer should be followed by a delivery by the seller:

$$\begin{aligned}
 &\mathbf{H}(\text{order}(\text{Customer}, \text{Seller}, \text{Good}), T_{\text{order}}) \\
 &\rightarrow \mathbf{E}(\text{do}(\text{Seller}, \text{deliver}(\text{Good})), T_{\text{delivery}}) \wedge T_{\text{delivery}} > T_{\text{order}}.
 \end{aligned} \tag{16}$$

Suppose that Alice placed an order, but in the meanwhile she was diagnosed a rare immunodeficiency, and she cannot meet people, except her family members. Her mother usually lives with her, but today she went out, so Alice locked the door, as it would be detrimental for her if any person got in the house. This prevents any delivery, but it is a minor issue for her compared to the consequences that Alice should face in case she met a stranger.

$$\begin{aligned}
 &\text{detrimental}(\text{deliver}(\text{Good}), \text{alice}). \\
 &\text{prevent}(\text{lockDoor}, \text{deliver}(\text{Good})).
 \end{aligned} \tag{17}$$

Given the following history

$$\begin{aligned}
 &\mathbf{H}(\text{order}(\text{alice}, \text{bob}, \text{computer}), 1) \\
 &\mathbf{H}(\text{do}(\text{alice}, \text{lockDoor}), 2)
 \end{aligned} \tag{18}$$

the expectation for Bob to deliver the good, raised by IC (16), is discharged by IC (12), because Alice performed an action that prevents the fulfillment.

4 $\text{SCIFF}^{\mathcal{D}}$ Declarative and Operational Semantics

4.1 Declarative Semantics

We now show how to deal with the discharge of expectations in the context of ALP. We first give an intuitive definition, then show its pitfalls and finally provide a correct definition.

In order to accept histories in which expectations may not have matching events, we need to extend the definition of fulfillment of expectations given in Eqs. (8) and (9); intuitively, a positive expectation is fulfilled if either there is a matching event or if the expectation has been discharged:

$$\begin{array}{l} \text{if } \mathbf{E}(\text{Content}, \text{Time}) \in \mathbf{EXP} \\ \text{then } \mathbf{H}(\text{Content}, \text{Time}) \in \mathbf{HAP} \vee \mathbf{D}(\mathbf{E}(\text{Content}, \text{Time})) \in \mathbf{EXP} \end{array} \quad (19)$$

and symmetrically for a negative expectation:

$$\begin{array}{l} \text{if } \mathbf{EN}(\text{Content}, \text{Time}) \in \mathbf{EXP} \\ \text{then } \mathbf{H}(\text{Content}, \text{Time}) \notin \mathbf{HAP} \vee \mathbf{D}(\mathbf{EN}(\text{Content}, \text{Time})) \in \mathbf{EXP} \end{array} \quad (20)$$

However in this way there could exist abductive answers with \mathbf{D} literals even if there is no explicit rule introducing them. For example, in the history of formula (15) an abductive answer would be¹

$$\begin{array}{l} \neg \mathbf{EN}(\text{rescind}(\text{alice}, \text{bob}, \text{buy}(\text{car}), \text{fixMileage}(\text{car}), i1, -), T2), \\ \mathbf{D}(\mathbf{EN}(\text{assertAgainst}(\text{alice}, -, i2), -)) \\ \mathbf{D}(\mathbf{E}(\text{do}(\text{alice}, \text{buy}(\text{car}), i1), -)) \\ \mathbf{D}(\mathbf{E}(\text{pay}(\text{alice}, \text{mechanic}, \text{lpgsystem}), -)) \\ \mathbf{NULL}(i1, i2) \end{array}$$

since it satisfies Eq. (5) (which takes into account knowledge base and integrity constraints), (6), (7), (19) and (20). Note that Alice is no longer required to pay the mechanic, because although no IC introduces explicitly the dischargement of the expectation that she should pay, the abductive semantics accepts the introduction of the literal $\mathbf{D}(\mathbf{E}(\text{pay}(\text{alice}, \text{mechanic}, \text{lpgsystem}), -))$.

We propose the following semantics.

Definition 3. *Abductive answer.*

If there is a set \mathbf{EXP} such that

1. *satisfies Eqs. (5), (6), and (7)*
2. *is minimal with respect to set inclusion within the sets satisfying point 3, considering only \mathbf{D} atoms; more precisely: there is no set \mathbf{EXP}' satisfying point 3 and such that $\mathbf{EXP}' \subset \mathbf{EXP}$ and $\mathbf{EXP} = \mathbf{EXP}' \cup F$, where F contains only \mathbf{D} atoms*
3. *satisfies Eqs. (19) and (20)*

then the set \mathbf{EXP} is an abductive answer, and we write $\langle KB, IC \rangle \models_{\mathbf{EXP}} \text{true}$.

¹ For brevity, we omit an expectation $\mathbf{E}(x)$ if we have already its discharged version $\mathbf{D}(\mathbf{E}(x))$.

4.2 Operational Semantics

Operationally, the proof procedure is extended with transition *Dischargement*.

If **EXP** contains two atoms $\mathbf{E}(x, T) \in \mathbf{PEND}$ and $\mathbf{D}(\mathbf{E}(x, T))$, then atom $\mathbf{E}(x, T)$ is moved to the set **FULF** of fulfilled expectations.

Similarly, if **EXP** contains two atoms $\mathbf{EN}(x, T) \in \mathbf{PEND}$ and $\mathbf{D}(\mathbf{EN}(x, T))$, then atom $\mathbf{EN}(x, T)$ is moved to the set **FULF** of fulfilled expectations.

Another modification is that transition *EN violation* is postponed after all other transitions (including the closure of the history). In fact, if we have $\mathbf{H}(x, 1)$, $\mathbf{EN}(x, T)$ and $\mathbf{D}(x, T)$, a failure would occur if the proof-procedure applied first *EN violation*. Instead, if *Dischargement* is applied first, expectation $\mathbf{EN}(x, T)$ is moved to the **FULF** set and transition *EN violation* is no longer applicable.

In case, at the end of a derivation, there are still expectations that are not fulfilled, the derivation is a failure derivation (and backtracking might occur to explore another branch, if available).

If a computation terminates with success, we write $\langle KB, \mathcal{IC} \rangle \vdash_{\mathbf{EXP}} \text{true}$.

4.3 Soundness

We are now ready to give the soundness statements; these statements rely on the soundness and completeness theorems of the **SCIFF** proof-procedure, so they hold in the same cases; for the **SCIFF** allowedness conditions over knowledge base and integrity constraints, we refer the reader to [5].

Theorem 1 (*Soundness of success*). *If $\langle KB, \mathcal{IC} \rangle$ is weakly **D**-allowed and $\langle KB, \mathcal{IC} \rangle \vdash_{\mathbf{EXP}} \text{true}$ then $\langle KB, \mathcal{IC} \rangle \models_{\mathbf{EXP}} \text{true}$.*

Proof. If no **D** atoms occur in \mathcal{IC} , the procedure coincides with **SCIFF**, which is sound [5]. In the case with **D** atoms in \mathcal{IC} , the procedure might report success in cases in which **SCIFF** reports failure due to the extended notion of fulfillment (Eqs. (19) and (20)). In such a case, the **D** atom must have been generated, and the only way to generate it is through an IC having such atom in the head (see Definition 1).

The procedure generates the atom only if the body of the IC is true. If the body is true, it means (from the soundness of **SCIFF**) that it is true also in the declarative semantics, so the **D** atom must be true also declaratively. In such a case, Eq. (19) (or (20)) is satisfied, meaning that the success was sound. \square

Theorem 2 (*Soundness of failure*). *If $\langle KB, \mathcal{IC} \rangle$ is weakly **D**-allowed and $\langle KB, \mathcal{IC} \rangle \not\models_{\mathbf{EXP}} \text{true}$, then $\exists \mathbf{EXP}' \subseteq \mathbf{EXP}$ such that $\langle KB, \mathcal{IC} \rangle \vdash_{\mathbf{EXP}'} \text{true}$.*

Proof. If there are no **D** atoms in \mathcal{IC} , the procedure coincides with **SCIFF**, so the completeness theorem of **SCIFF** holds [5]. In the case with **D** atoms in \mathcal{IC} , the declarative semantics allows as abductive answers some sets that would not have been returned by **SCIFF**, and in which expectations are not fulfilled by actual events, but discharged through an abduced **D** atom.

Consider such an abductive answer **EXP**. We prove by contradiction that each **D** atom in **EXP** occurs in the head of an IC whose body is true. In fact, if

a **D** atom in **EXP** was not in the head of an IC whose body is true, then the set **EXP'** obtained by removing the **D** atom from **EXP** would satisfy Eq. (5). On the other hand, since **EXP** satisfies Eqs. (6) and (7) and those equations do not involve **D** atoms, also **EXP'** satisfies those equations. This means that **EXP** does not satisfy condition 2 of Definition 3, which means that **EXP** was not an abductive answer and we get a contradiction.

Since each **D** atom occurs in the head of an IC whose body is true, the procedure applies such IC and abduces that atom. This means that the corresponding expectation becomes discharged, and hence it does not cause failure. \square

5 Related Work

Many authors have investigated legal and normative applications of deontic logics. The use of such logics was initially debated when taking into account permissions. For example, in [16] the authors present an approach based on input/output logics [32] for formalizing conditional norms, obligations and permissions in a scenario where many hierarchically organized authorities are present. In such a scenario, there can be norms that are more important than others and therefore the authors consider a hierarchy of norms, defined by “meta-norms”, and different types of permissions with different strengths. However, the focus of [16] is on helping the legislator to understand how the modification of a norm or the definition of a new one may change the whole normative system. In fact, following the input/output logic’s semantics, [16] is not concerned about the truth value of formulae representing (part of) norms but defines a cause-effect link between inputs and obligatory outputs in an abductive-like way.

Recently deontic logics have been increasingly applied to legal domains. In [26] the authors discuss the impact new contracts, which introduce new constraints, may have on already existing business processes. The authors present a logic called FCL (Formal Contract Language), based on RuleML, for representing contracts in a formal way. The language allows automatic checking and debugging, analysis and reasoning [25]. In [26] a normal form of FCL, called NFCL, is presented with the aim of having a clean, complete and non-redundant representation of a contract. This normal form is obtained by merging the new constraints with the existing ones and cleaning up the redundancies by using the notion of subsumption. The result points out possible conflicts among contracts and how each contract is intertwined with the whole business process. Similar results can be accomplished with *SCIFF^D* which allows checking the consistency of the *SCIFF^D* program representing the constraints of the business process.

A different approach is the combination of temporal logics with deontic logics. An example is given in [1], where the authors define Normative Temporal Logic (NTL) which replaces the standard operators of the well-known CTL [22] with deontic operators. The use of time, which forces the sequentiality of the constraints, avoids many paradoxes typical of standard deontic logic, such as those involving contrary-of-duty. Moreover, the authors present the Simple Reactive Modules Language (SRML) which follows NTL and allows the execution of

model checking in four different scenarios depending on the presence or absence of an interpretation of the normative system and on the definition of the model under examination. Similarly, SCIFF^D can manage time although it does not follow the temporal logic semantics. A similar approach is proposed by the same authors in [2] where they present the Norm Compliance CTL (NCCTL). This logic extends CTL by adding a new deontic-like operator P modeling coalitions between norms which cooperate in the normative system. NCCTL is equipped with a model checker, called NORMC [31]. Since SCIFF performs on-the-fly checking of compliance, the two systems cannot be directly compared.

The **AD** system [9] is a deontic logic that supports defeasible obligations by means a revision operator (called f), which represents the assumptions that normally come with an explicitly stated condition. Intuitively, $fA \Rightarrow O(B)$ means that A implies that B is obligatory, as long as the usual assumptions about A are true. The SCIFF^D semantics implements an implicit assumption that an expectation is not discharged (and is therefore required to be fulfilled), which can be defeated by an explicit discharge atom (which allows for the expectation not to be fulfilled).

This paper shows that representing and reasoning with norms facilitate for the adoption of such approaches in many scenarios. Checking whether certain facts are compliant with the normative system in use is in fact often needed. Abductive frameworks such as SCIFF^D can also be used, for example, in forensics for analysing and arguing on evidence of a crime or for explaining causal stories, sequence of states and events forming (part of) a case. Such stories must be coherent and there must be a process, usually abductive, able to prove their truthfulness. These necessities are pointed out for example in [13,14], where the authors present a hybrid framework which combines the two most used approaches in reasoning about criminal evidences: argumentative and story-based analysis. Both of them could benefit from the use of normative systems.

6 Conclusions

In this article we continue our line of research that applies abductive logic programming to the formalization of normative systems.

We introduced the SCIFF^D language, extending the SCIFF abductive framework with the notion of dischargeable obligation. Dischargeable obligations can occur in the head of forward rules (named ICs), fired under specific conditions mentioned in the body of the rules. The SCIFF^D declarative semantics and its operational counterpart for verification accordingly extend SCIFF 's, and soundness is proved under syntactic conditions over these (discharging) constraints.

To experiment the framework we considered case studies requiring the notion of discharging of an obligation. In particular, we considered the articles in the Japanese Civil Code that deal with the rescission of manifestation of intentions and prevention of fulfillment of conditions. We also show - informally - the result of running the operational support upon this example in some simple scenarios.

Acknowledgements. This work was partially supported by the “GNCS-INdAM”.

References

1. Ågotnes, T., van der Hoek, W., Rodríguez-Aguilar, J.A., Sierra, C., Wooldridge, M.: On the logic of normative systems. In: Veloso, M.M. (ed.) 20th International Joint Conference on Artificial Intelligence, Hyderabad, India (IJCAI 2007), vol. 7, pp. 1175–1180. AAAI Press, Palo Alto (2007)
2. Ågotnes, T., van der Hoek, W., Wooldridge, M.: Robust normative systems and a logic of norm compliance. *Log. J. IGPL* **18**(1), 4–30 (2010)
3. 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 8th Symposium on Principles and Practice of Declarative Programming, pp. 39–50. ACM Press, New York, July 2006
4. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. *Appl. Artif. Intell.* **20**(2–4), 133–157 (2006)
5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM T. Comput. Log.* **9**(4), 29:1–29:43 (2008)
6. Alberti, M., Gavanelli, M., Lamma, E.: *Deon*⁺: abduction and constraints for normative reasoning. In: Artikis, A., Craven, R., Kesim Çiçekli, N., Sadighi, B., Stathis, K. (eds.) Logic Programs, Norms and Action. LNCS, vol. 7360, pp. 308–328. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-29414-3_17](https://doi.org/10.1007/978-3-642-29414-3_17)
7. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., Torroni, P.: Mapping deontic operators to abductive expectations. *Comput. Math. Organ. Th.* **12**(2–3), 205–225 (2006)
8. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interactions using social integrity constraints. *Electr. Notes Theor. Comput. Sci.* **85**(2), 94–116 (2003)
9. Alchourrón, C.E.: Detachment and defeasibility in deontic logic. *Studia Logica* **57**(1), 5–18 (1996)
10. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: partial meet contraction and revision functions. *J. Symbolic Logic* **50**(2), 510–530 (1985)
11. ALFEBIITE: A Logical Framework for Ethical Behaviour Between Infohabitants in the Information Trading Economy of the Universal Information Ecosystem. IST-1999-10298 (1999)
12. Arisha, K.A., Ozcan, F., Ross, R., Subrahmanian, V.S., Eiter, T., Kraus, S.: IMPACT: a platform for collaborating agents. *IEEE Intell. Syst.* **14**(2), 64–72 (1999)
13. Bex, F., Prakken, H., Reed, C., Walton, D.: Towards a formal account of reasoning about evidence: argumentation schemes and generalisations. *Artif. Intell. Law* **11**(2–3), 125–165 (2003)
14. Bex, F.J., van Koppen, P.J., Prakken, H., Verheij, B.: A hybrid formal theory of arguments, stories and criminal evidence. *Artif. Intell. Law* **18**(2), 123–152 (2010)
15. Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. *Comput. Math. Organ. Th.* **12**, 71–79 (2006)
16. Boella, G., van der Torre, L.W.N.: Permissions and obligations in hierarchical normative systems. In: Zelezniokow, J., Sartor, G. (eds.) 9th International Conference on Artificial Intelligence and Law (ICAIL 2003), Edinburgh, Scotland, UK, Proceedings, pp. 109–118. ACM Press (2003)

17. Bracciali, A., et al.: The KGP model of agency for global computing: computational model and prototype implementation. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS, vol. 3267, pp. 340–367. Springer, Heidelberg (2005). doi:[10.1007/978-3-540-31794-4_18](https://doi.org/10.1007/978-3-540-31794-4_18)
18. Dignum, V., Meyer, J.J., Weigand, H.: Towards an organizational model for agent societies using contracts. In: Castelfranchi, C., Lewis Johnson, W. (eds.) 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), Part II, pp. 694–695. ACM Press (2002)
19. Dignum, V., Meyer, J.J., Weigand, H., Dignum, F.: An organizational-oriented model for agent societies. In: 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002). ACM Press (2002)
20. Eiter, T., Subrahmanian, V., Pick, G.: Heterogeneous active agents, I: semantics. *Artif. Intell.* **108**(1–2), 179–255 (1999)
21. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. *J. ACM* **42**(1), 3–42 (1995)
22. Emerson, E.A.: Temporal and modal logic. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pp. 995–1072. Elsevier (1990)
23. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. *J. Logic Program.* **33**(2), 151–165 (1997)
24. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: An abductive framework for Datalog+− ontologies. In: De Vos, M., Eiter, T., Lierler, Y., Toni, F. (eds.) *Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015)*, No. 1433 in CEUR-WS, Sun SITE Central Europe, Aachen, Germany (2015)
25. Governatori, G.: Representing business contracts in RuleML. *Int. J. Coop. Inf. Syst.* **14**(2–3), 181–216 (2005)
26. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance checking between business processes and business contracts. In: 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Hong Kong, China, pp. 221–232. IEEE Computer Society (2006)
27. Governatori, G., Rotolo, A.: BIO logical agents: norms, beliefs, intentions in defeasible logic. *Auton. Agent Multi Ag.* **17**(1), 36–69 (2008)
28. Jaffar, J., Maher, M.: Constraint logic programming: a survey. *J. Logic Program.* **19–20**, 503–582 (1994)
29. Japanese Civil Code, Part I. https://en.wikisource.org/wiki/Civil_Code_of_Japan/Part.I. Accessed 19 July 2016
30. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. *J. Logic Comput.* **2**(6), 719–770 (1993)
31. Kazmierczak, P., Pedersen, T., Ågotnes, T.: NORMC: a norm compliance temporal logic model checker. In: Kersting, K., Toussaint, M. (eds.) 6th Starting AI Researchers’ Symposium (STAIR 2012), FRONTIERS, Montpellier, France, vol. 241, pp. 168–179. IOS Press (2012)
32. Makinson, D., van der Torre, L.W.N.: Constraints for input/output logics. *J. Philos. Logic* **30**(2), 155–185 (2001)
33. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. *J. Appl. Non Classical Logics* **7**(1), 25–75 (1997)
34. Sadri, F., Stathis, K., Toni, F.: Normative KGP agents. *Comput. Math. Organ. Th.* **12**(2–3), 101–126 (2006)
35. 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)
36. Wright, G.: Deontic logic. *Mind* **60**, 1–15 (1951)

Rules and Reasoning

International Joint Conference, RuleML+RR 2017,

London, UK, July 12-15, 2017, Proceedings

Costantini, S.; Franconi, E.; Van Woensel, W.;

Kontchakov, R.; Sadri, F.; Roman, D. (Eds.)

2017, XVIII, 239 p. 43 illus., Softcover

ISBN: 978-3-319-61251-5