An Embedding of Input-output Logic in Deontic Logic Programs

Ricardo Gonçalves and José Júlio Alferes[*]

CENTRIA - Dep. Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Abstract. Parametrized logic programs, for which a syntax and natural declarative semantics have been recently defined, are very expressive logic programs under the stable model semantics (also usually called answer set programs) in which complex formulas of a given parameter logic are allowed to appear in the body and head of rules. The choice of the parameter logic depends largely on the domain of the problem to be modeled.
In this paper we show how input-output logic can be embedded into parametrized logic programs, by choosing deontic logic as the parameter logic. This embedding not only shows how to recast input-out logic in this extension of answer set programming, but also sheds light on how to extend input-output logic with some interesting non-monotonic features.

1 Introduction

Deontic logic is well-known to be a fundamental tool for modeling normative reasoning. Since the seminal work of von Wright [16] many have investigated and developed systems of deontic logic. One such system is the modal logic KD, usually known as Standard Deontic Logic (SDL) [2]. Although accepted as a tool for modeling normative assertions, SDL has shown not to be enough for the task of representing norms [3]. First of all, it became clear that the classical implication of SDL does not provide a faithful representation for the conditional obligations that usually populate a normative system. Moreover, SDL is unable to deal with some paradoxes, namely the so-called contrary-to-duty paradoxes. Contrary-to-duty paradoxes encode the problem of what obligations should follow from a normative system in a situation where some of the existing obligations are already being violated.

These limitations of SDL fostered the development of several approaches modeling conditional obligations in such a way that they have a more reasonable behavior in the face of the aforementioned paradoxes [15, 9, 11, 1, 14].

Input-output logic [11] takes its origins precisely in the study of conditional obligations. Input-output logic uses a rule-based representation of conditional norms. A conditional norm is represented as a pair $\langle\varphi, \psi\rangle$where φ and ψ are formulas. Its intuitive reading is that the body φ is thought of as an input, representing some condition

[*] The first author was supported by FCT under the postdoctoral grant SFRH/BPD/47245/2008. The work was partially supported by project ERRO, PTDC/EIA-CCO/121823/2010. We'd like to thank the reviewers who, by their constructive comments, really helped to improve the paper.

or situation, and the head ψ is thought of as an output, representing what is obligatory in that situation.

As any other approach to normative reasoning, input-output logic should reasonably deal with contrary-to-duty situations. In [10], ideas from non-monotonic reasoning were used to extend input-output logic to cope with contrary-to-duty situations.

Parametrized logic programming [5] was introduced as an extension of answer set programming [4] with the motivation of providing a meaning to theories combining both logic programming connectives with other logical connectives, and allowing complex formulas using these connectives to appear in the head and body of a rule. The main idea is to fix a monotonic logic $\mathcal{L}$, called the parameter logic, and build up logic programs using formulas of $\mathcal{L}$ instead of just atoms. The obtained parametrized logic programs have, therefore, the same structure of normal logic programs, being the only difference the fact that atomic symbols are replaced by formulas of $\mathcal{L}$.

When applying this framework, the choice of the parameter logic depends on the domain of the problem to be modeled. As examples, [5] shows how to obtain the answer-set semantics, a paraconsistent version of it, and also the semantics of MKNF hybrid knowledge bases [13], using an appropriate choice of the parameter logic.

Parametrized logic programming can be seen as a framework which allows to add non-monotonic rule based reasoning on top of an existing (monotonic) language. This view is quite interesting, in particular in those cases where we have already a monotonic logic to model a problem, but we are still lacking some conditional or non-monotonic reasoning. In these situations, parametrized logic programming offers a modular framework for adding such conditional and non-monotonic reasoning, without having to give up of the monotonic logic at hand. One interesting example is the case of MKNF hybrid knowledge bases, where the existing monotonic logics are the description logics.

In this paper, after presenting some background on input-output logic (Section 2) and parametrized logic programming (Section 3), we propose the use of standard deontic logic as the parameter of this general non-monotonic parametrized logic programming framework (Section 4) to obtain a very expressive language - deontic logic programs - along with a purely declarative semantics. We then show (Section 5) that this language allows to represent and reason about norms, including dealing with contrary-to-duty situations, and show that it is expressive enough to embed input-output logic. With the help of an example, we shed light on how deontic logic programs in fact extend input-output logic.

2 Input-Output Logic

The key idea in input-output logic (IO logic) [10] is to represent norms using pairs of formulas, rather then with just formulas, as it is usual in deontic logics. The central elements in the language of IO logic are, therefore, the pairs $\langle\varphi, \psi\rangle$, where φ and ψ are classical propositional formulas. Intuitively, a pair $\langle\varphi, \psi\rangle$ represents the conditional norm that whenever the body φ (the input) is true then the head ψ (the output) is obligatory. As an example, the pair

$$\langle driving \wedge redSignal, stop\rangle$$

can be seen as the representation of the norm stating that, whenever you are driving and there is a red signal, you have the obligation to stop.

Definition 1. *A generating set is a set G of pairs.*

Generating sets can be seen as the formal representation of a normative code, i.e., a set of conditional norms. The term generating set comes from the intuition that it generates the output from a given input. Given a generating set G and a set A of propositional formulas, we consider the set

$$G(A) = \{\psi : \langle \varphi, \psi \rangle \in G \text{ and } \varphi \in A\}.$$

Intuitively, the set $G(A)$ can be seen as the direct consequences of the normative system G given a set of facts A. The construction of the set $G(A)$ does not have into account the logical interdependence between formulas. For example, if $G = \{\langle p, q \rangle\}$ and $A = \{p \wedge r\}$ then, we have that $q \notin G(A)$.

The semantics of IO logic is an operational semantics which is parametrized by the choice of the so-called out operations. These out operations represent the different ways in which the logical interdependence between formulas can be handled.

Operation $out(G, A)$ takes a generating set G and a (input) set of formulas A and returns a (output) set of formulas. Four natural out operations are usually considered: the simple-minded operator out_1, the basic operator out_2, reusable operator out_3, and reusable basic operator out_4.

In this paper we focus on two of these operators[1]: out_1 and out_3. Given a set A of formulas we denote by $Cn(A)$ the set of consequence of A in classical logic. Recall that a classical theory is a set T such that $Cn(T) = T$. We can now define the out operations.

Definition 2. *Given a generating set G and a set A of propositional formulas:*

- $out_1(G, A) = Cn(G(Cn(A)))$
- $out_3(G, A) = \bigcap\{Cn(G(B)) : A \subseteq B,\ B = Cn(B), and\ G(B) \subseteq B\}$

Moreover, for each operator out_n, we can consider out_n^+, the correspondent throughput operator that allows inputs to reappear as outputs. These operators are defined as

$$out_n^+(G, A) = out_n(G \cup Id, A)$$

where the Id is the identity binary relation, i.e., is defined as

$$Id = \{\langle \varphi, \varphi \rangle : \varphi \text{ classical formula}\}.$$

In what follows, we use $out(G, A)$ when referring to any of the above out operations.

Although the above formulation of input-output logic already gives a reasonable account of conditional obligations, it is not enough for reasoning with contrary-to-duty situations. Contrary-to-duty situations encode the problem of what obligations should follow from a normative system in a situation where some of the existing obligations

[1] We do not consider all 4 operators due to lack of space, and also because the other two are less interesting: $out_2^+ = out_4^+$ and they both degenerate into classical logic.

are already being violated. Contrary-to-duty situations were called paradoxical only because SDL failed to give them a reasonable account. They are, in fact, very common in a normative scenario. The norms of a normative system should not be seen as hard constraints, i.e., the obligations in a normative system can be violated and, in those cases, the normative system should also specify what sanctions follow from these violations.

In order to cope with contrary-to-duty paradoxes, IO logic was extended in [11]. There, ideas from non-monotonic reasoning were used to deal with problems related with consistency. The issue was how to deal with excessive output, i.e., those cases in which the output was itself inconsistent or it was inconsistent with respect to the input. In the last case the input set is said to be inconsistent with the output. The strategy to overcome this problem was to cut back the set of generators just below the threshold of yielding an excessive output. The following general notions of maxfamily and outfamily were introduced precisely to deal with excessive output.

Given a generating set G and a set A of propositional formulas, $maxfamily(G, A)$ is the set of maximal subsets of G for which the output operator yields a set consistent with the input, i.e., the set

$$\{H : H \subseteq G \text{ and } H \text{ is maximal s.t. } out(H, A) \text{ is consistent with A}\}.$$

The set $outfamily(G, A)$ collects the outcomes of each element in $maxfamily(G, A)$:

$$outfamily(G, A) = \{out(H, A) : H \in maxfamily(G, A)\}.$$

Recall that for the operations admitting throughput, namely out_n^+, we have that $A \subseteq out_n^+(G, A)$. Therefore, for those output operators it is equivalent to say that $out_n^+(G, A)$ is consistent with A and that $out_n^+(G, A)$ is itself consistent.

3 Parametrized Logic Programming

In this section we introduce the syntax and semantics of normal parametrized logic programs [5]. The syntax of these rich logic program has the same structure of normal logic programs. The key difference is that the atomic symbols of a normal parametrized logic program are replaced by formulas of a parameter logic.

First of all we introduce the necessary concepts related with the notion of (monotonic) logic.

Definition 3. *A (monotonic) logic is a pair $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ where L is a set of formulas and $\vdash_{\mathcal{L}}$ is a Tarskian consequence relation [17] over L, i.e. satisfying the following conditions, for every $T \cup \Phi \cup \{\varphi\} \subseteq L$,*

Reflexivity: *if $\varphi \in T$ then $T \vdash_{\mathcal{L}} \varphi$;*
Cut: *if $T \vdash_{\mathcal{L}} \varphi$ for all $\varphi \in \Phi$, and $\Phi \vdash_{\mathcal{L}} \psi$ then $T \vdash_{\mathcal{L}} \psi$;*
Weakening: *if $T \vdash_{\mathcal{L}} \varphi$ and $T \subseteq \Phi$ then $\Phi \vdash_{\mathcal{L}} \varphi$.*

When clear from the context we write $\vdash$ instead of $\vdash_{\mathcal{L}}$. Let $Th(\mathcal{L})$ be the set of *theories of* $\mathcal{L}$, i.e. the set of subsets of L closed under the relation $\vdash_{\mathcal{L}}$. It is well-known that, for every (monotonic) logic $\mathcal{L}$, the tuple $\langle Th(\mathcal{L}), \subseteq \rangle$ is a complete lattice with smallest element the set $Theo = \emptyset^{\vdash}$ of theorems of $\mathcal{L}$ and the greatest element the set

L of all formulas of $\mathcal{L}$. Given a subset A of L we denote by $A^{\vdash}$ the smallest theory that contains A. $A^{\vdash}$ is also called the theory generated by A.

In what follows we consider fixed a (monotonic) logic $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ and call it the *parameter logic*. The formulas of $\mathcal{L}$ are dubbed *(parametrized) atoms* and a *(parametrized) literal* is either a parametrized atom φ or its negation $not\ \varphi$, where as usual not denotes negation as failure. We dub *default literal* those of the form $not\ \varphi$.

Definition 4. *A* normal $\mathcal{L}$-parametrized logic program *is a set of rules*

$$\varphi \leftarrow \psi_1, \dots, \psi_n, not\ \delta_1, \dots, not\ \delta_m$$

where $\varphi, \psi_1, \dots, \psi_n, \delta_1, \dots, \delta_m \in L$.

A definite $\mathcal{L}$-parametrized logic program *is a set of rules without negations as failure, i.e. of the form* $\varphi \leftarrow \psi_1, \dots, \psi_n$ *where* $\varphi, \psi_1, \dots, \psi_n \in L$.

We now present the stable model like semantics [4] of these very expressive logic programs. In the traditional definition of a stable model semantics, an interpretation is taken to be just a set of atoms. Following naively this idea in the case of a parametrized logic program, and since the atoms are now formulas of the parameter logic, we could think of considering as interpretations any set of formulas of the parameter logic. This idea, however, does not work. The problem is that, contrary to the case of atoms, the parametrized atoms are not independent of each other. Consider, just as an example, the case where the parameter logic is classical propositional logic (CPL). Then, if an interpretation contains $p \wedge q$ then it should also contain both p and q. This interdependence between the parametrized atoms is governed by the consequence relation of the parameter logic. Returning to the example with CPL, it is well-known that $p \wedge q \vdash_{CPL} p$ and $p \wedge q \vdash_{CPL} q$. To account for this interdependence, the key idea is to use logical theories as interpretations. Recall that a logical theory of a logic is a set of formulas closed under the consequence of the logic. Therefore, returning to the example of CPL, if an interpretation I contains $p \wedge q$ and it also contains both p and q, since $p \wedge q \vdash_{CPL} p$ and $p \wedge q \vdash_{CPL} q$ and I is closed under logical consequence.

Definition 5. *A (parametrized) interpretation is a theory of* $\mathcal{L}$.

As usual, an interpretation T can be seen as a tuple $\langle T, F \rangle$ where F is the complement, wrt L, of T. Note that, defined as such, F is not a theory, viz. it is not closed under the consequence of the logic. E.g. F does not, and should not, include tautologies of $\mathcal{L}$.

The usual ordering defined over interpretations can easily be generalised.

Definition 6. *If* I *and* J *are two interpretations then we say that* $I \leq J$ *if* $I \subseteq J$.

Given the above ordering, the notions of minimal and least interpretations are defined in the usual way.

Definition 7. *An interpretation* I *satisfies a rule* $\varphi \leftarrow \psi_1, \dots, \psi_n, not\ \delta_1, \dots, not\ \delta_m$ *if* $\varphi \in I$ *whenever* $\psi_i \in I$ *for every* $i \in \{1, \dots, n\}$*, and* $\delta_j \notin I$ *for every* $j \in \{1, \dots, m\}$.

If an interpretation I satisfies a rule r we also say that I is closed under r.

Definition 8. *An interpretation is a model of a $\mathcal{L}$-parametrized logic program P if it satisfies every rule of P. We denote by $Mod^{\mathcal{L}}(P)$ the set of models of P.*

As usual, we start by defining the semantics of definite programs.

Definition 9. *The stable model semantics of a definite $\mathcal{L}$-parametrized logic program P is its least model $S_P^{\mathcal{L}}$.*

In order to guarantee that the above notion is well-defined, in [5] it was proved that every definite $\mathcal{L}$-parametrized logic program P has a least model. This least model is precisely the intersection of all models of P, i.e., $S_P^{\mathcal{L}} = \bigcap_{I \in Mod^{\mathcal{L}}(P)} I$.

To define the stable model semantics of a $\mathcal{L}$-parametrized logic programs with negation as failure a Gelfond-Lifschitz like operator is used.

Definition 10. *Let P be a normal $\mathcal{L}$-parametrized logic program and I an interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:*

- *remove from P all rules which contain a literal $not\ \varphi$ such that $I \vdash_{\mathcal{L}} \varphi$;*
- *remove from the remaining rules all default literals.*

Since $\frac{P}{I}$ is a definite $\mathcal{L}$-parametrized program, it has an unique least model J. We define $\Gamma(I) = J$.

A stable model is then defined as a fixed point of this operator.

Definition 11. *An interpretation I of a $\mathcal{L}$-parametrized logic program P is a stable model of P iff $\Gamma(I) = I$. A formula φ is true under the stable model semantics iff it belongs to all stable models of P.*

4 Deontic Logic Programs

The choice of the parameter logic in the parametrized logic approach depends on the domain of the problem to be modeled. For representing normative systems we use standard deontic logic as the parameter logic, thus obtaining the deontic logic programs. We start by briefly recalling standard deontic logic (SDL) – see [2] for further details – and then we introduce the deontic logic programs.

4.1 Standard deontic logic

The formal study of deontic logic was highly influenced by modal logic. In fact, SDL, which has emerged as the standard system for deontic reasoning, is a modal logic with two modal operators, one for obligation and another for permission.

Formally, the language of SDL, dubbed L_{SDL}, is constructed from a set $Prop$ of propositional symbols using the usual classical connectives $\neg$, $\Rightarrow$, and the unary deontic operator $\mathbf{O}$ (obligation). The classical connectives $\vee$ and $\wedge$ can be defined, as usual, as abbreviations. The permission operator can be defined as an abbreviation $\mathbf{P} := \neg\mathbf{O}\neg$.

The semantics of SDL is a Kripke-style semantics. A Kripke model is a tuple $\langle W, R, \mathcal{V} \rangle$, where W is a set, the possible worlds, $R \subseteq W \times W$ is the accessibility relation, and $\mathcal{V} : W \to 2^{Prop}$ is a function assigning, to each world, the set of propositional symbols true at that world. We assume that R is a serial relation, i.e., for every $w \in W$ there exists $w' \in W$ such that wRw'. We define the satisfaction of a formula φ in a model $\mathcal{M} = \langle W, R, \mathcal{V} \rangle$ at a world $w \in W$ by induction on the structure of φ.

i) $\mathcal{M}, w \Vdash p$ if $p \in \mathcal{V}(w)$, for $p \in Prop$;
ii) $\mathcal{M}, w \Vdash \neg\varphi$ if $\mathcal{M}, w \nVdash \varphi$;
iii) $\mathcal{M}, w \Vdash \varphi_1 \Rightarrow \varphi_2$ if $\mathcal{M}, w \nVdash \varphi_1$ or $\mathcal{M}, w \Vdash \varphi_2$;
iv) $\mathcal{M}, w \Vdash \mathbf{O}(\varphi)$ if $\mathcal{M}, w' \Vdash \varphi$ for every w' s.t. $\langle w, w' \rangle \in R$.

We say that an SDL formula φ is a *logical consequence* of a set Φ of SDL formulas, written $\Phi \vdash_{SDL} \varphi$, if for every Kripke model $\mathcal{M} = \langle W, R, \mathcal{V} \rangle$ and every world $w \in W$ we have that $\mathcal{M}, w \Vdash \varphi$ whenever $\mathcal{M}, w \Vdash \delta$ for every $\delta \in \Phi$. A formula φ is said to be a SDL *theorem* if $\emptyset \vdash_{SDL} \varphi$.

Before we continue we need to make clear one important point. We are using here the so-called local consequence relation, contrasted with the global consequence relation defined as $\Phi \vdash_g \varphi$ if for every Kripke model $\mathcal{M} = \langle W, R, \mathcal{V} \rangle$ we have that $\mathcal{M}, w \Vdash \varphi$ for every world $w \in W$ whenever $\mathcal{M}, w \Vdash \delta$ for every world $w \in W$ and every $\delta \in \Phi$. Although the local and the global consequence relations are quite different, this difference is sometimes neglected. The reason for this confusion is the fact that the set of theorems is the same for both consequences. If one has only interest in the set of theorems, then it is irrelevant which consequence it uses. But, if we are interested in the consequence relation, then this distinction should be made. In our approach, since the consequence relation is a fundamental tool, we do not neglect this difference and work with the local consequence which is more adequate for normative reasoning. In fact, from our point of view, the global consequence does not faithfully represent normative reasoning. For example, we have that $\varphi \vdash_g \mathbf{O}(\varphi)$ but this is not a valid reasoning if $\mathbf{O}$ is to be read as an obligation.

We now define the notion of logical theory. It will play a fundamental role in the definition of the semantics for deontic logic programs.

Definition 12. *A set of SDL formulas Φ is said to be a SDL logical theory if Φ is closed under SDL consequence, i.e., for every $\varphi \in L_{SDL}$ if $\Phi \vdash_{SDL} \varphi$ then $\varphi \in \Phi$.*

We denote by $Th(SDL)$ the set of theories of SDL. We recall that $\langle Th(SDL), \subseteq \rangle$ is a complete lattice with smallest element the set $Theo(SDL)$ of theorems of SDL and greatest element the set L_{SDL} of all SDL formulas. Given a subset A of L_{SDL} we denote by $A^{\vdash_{SDL}}$ the smallest SDL theory that contains A.

Although well-known, we stress that the use of complex propositional formulas inside the deontic operators is strictly necessary to represent several kinds of reasoning. For example, $\{\mathbf{P}(\varphi), \mathbf{O}(\varphi \Rightarrow \psi)\} \vdash_{SDL} \mathbf{P}(\psi)$, but $\{\mathbf{P}(\varphi), \mathbf{O}(\varphi) \Rightarrow \mathbf{O}(\psi)\} \nvdash_{SDL} \mathbf{P}(\psi)$. Other interesting consequences in SDL are $\{\varphi, \mathbf{O}(\varphi \Rightarrow \psi)\} \nvdash_{SDL} \mathbf{O}(\psi)$, also $\{\mathbf{O}(\varphi), (\varphi \Rightarrow \psi)\} \nvdash_{SDL} \mathbf{O}(\psi)$, but $\{\mathbf{O}(\varphi), \mathbf{O}(\varphi \Rightarrow \psi)\} \vdash_{SDL} \mathbf{O}(\psi)$.

4.2 Deontic Logic Programs

In this section we introduce deontic logic programs, which are obtained from the general framework introduced in Section 3, taking SDL as the parameter logic.

Definition 13. *A deontic logic program is a set of ground rules*

$$\varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \delta_1, \ldots, not\ \delta_m$$

where $\varphi, \psi_1, \ldots, \psi_n, \delta_1, \ldots, \delta_m \in L_{SDL}$.

Note that, contrarily to some works in the literature on the combination of non-monotonic reasoning and deontic logic, for example [7, 10, 8], deontic formulas can appear both in the head and in the body of a rule, and moreover, they can be complex formulas and not just atomic formulas. This extra flexibility is relevant, for example, to deal with non-compliance and application of sanctions. We can use the rule [2]

$$\mathbf{O}(payFine(X)) \leftarrow \mathbf{O}(pay(X)), not\ pay(X)$$

to express that if an agent has the obligation to pay some bill, and it is not known that the agent has payed it, then the agent is obliged to pay a fine.

We should stress again that deontic logic programs allow to represent rules with complex deontic formulas. This is fundamental to represent several different deontic situations. Consider for example, two normative systems (modeled as deontic logic programs): $N_1 = \{\mathbf{O}(p) \Rightarrow \mathbf{O}(q) \leftarrow\}$ and $N_2 = \{\mathbf{O}(p \Rightarrow q) \leftarrow\}$. These normative systems express quite different deontic information. In fact, if we assume that p and $\neg q$ are the case then, in the case of N_1 we do not have a violation of an obligation, whereas in the case of N_2 we do have a violation.

5 Embedding Input-Output Logic

In this section we present an embedding of IO logic in deontic logic programs. The results presented here can be seen as a strengthening of the existing weak connection drawn in [10] between input-output logic and Reiter's default logic.

Recall that $S_{\mathcal{P}}$ denotes the unique stable model of a definite deontic program $\mathcal{P}$ and $SM(\mathcal{P})$ the set of stable models of a deontic logic program $\mathcal{P}$.

The following lemma shows that, given a generating set G and a set A of formulas, we can define, for each *out* operation, a deontic logic program whose stable model semantics captures the operational semantics given by the respective *out* operator.

Lemma 1. *Let* G *be a generating set,* A *an input set consistent with the output, and let* φ *and* ψ *stand for classical propositional formulas.*

[2] In this rule we abuse notation and use a variable. As usual in answer-set programming, this is to be understood as a macro, standing for all possible ground instances of the rule.

1) Let $\mathcal{P}_1 = \{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\} \cup$
$\{\varphi \leftarrow : \varphi \in A\}$.

Then, $out_1(G, A) = \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_1}\}$.

2) Let $\mathcal{P}_3 = \{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\} \cup$
$\{\varphi \leftarrow : \varphi \in A\} \cup$
$\{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\}$.

Then, $out_3(G, A) = \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\}$.

3) Let $\mathcal{P}_1^+ = \{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\} \cup$
$\{\varphi \leftarrow : \varphi \in A\} \cup$
$\{\mathbf{O}(\varphi) \leftarrow : \varphi \in A\}$.

Then, $out_1^+(G, A) = \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_1^+}\}$.

4) Let $\mathcal{P}_3^+ = \{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\} \cup$
$\{\mathbf{O}(\psi) \leftarrow \mathbf{O}(\varphi) : \langle\varphi, \psi\rangle \in G\} \cup$
$\{\mathbf{O}(\varphi) \leftarrow : \varphi \in A\} \cup$
$\{\varphi \leftarrow : \varphi \in A\}$.

Then, $out_3^+(G, A) = \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3^+}\}$.

Proof. Let us prove the result for out_1 and out_3. The other cases can be proved similarly. For each case we prove the two inclusions.

Case 1). We start by proving that $out_1(G, A) \subseteq \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_1}\}$. First of all, note that $Cn(A) \subseteq S_{\mathcal{P}_1}$ because $\{\varphi \leftarrow : \varphi \in A\} \subseteq \mathcal{P}_1$ and $S_{\mathcal{P}_1}$ is a SDL-theory. Therefore, we can conclude that $G(Cn(A)) \subseteq \{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\}$ since $S_{\mathcal{P}_1}$ is closed under the rules of $\mathcal{P}_1$, in particular those of the form $\{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\}$. We then have that $Cn(G(Cn(A))) \subseteq Cn(\{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\})$. But it is easy to prove that $\{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\}$ is a CPL-theory, given the fact that $S_{\mathcal{P}_1}$ is a SDL-theory. Therefore, $Cn(G(Cn(A))) \subseteq Cn(\{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\}) = \{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\}$.
We now prove that $\{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_1}\} \subseteq out_1(G, A)$. Let $\Phi = (\{\mathbf{O}(\psi) : \psi \in out_1(G, A)\} \cup A)^{\vdash_{SDL}}$ be a SDL-theory. Clearly, Φ is a model of $\mathcal{P}_1$. Therefore, $S_{\mathcal{P}_1} \subseteq \Phi$ because $S_{\mathcal{P}_1}$ is the smallest model of $\mathcal{P}_1$. Then, clearly we have that $\{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\} \subseteq \{\psi : \mathbf{O}(\psi) \in \Phi\}$. But, it is easy to prove that $\{\psi : \mathbf{O}(\psi) \in \Phi\} = out_1(G, A)$.

Case 2). We start by proving that $out_3(G, A) \subseteq \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\}$. Let Φ be a model of $\mathcal{P}_3$. Consider the set Φ_{CPL} of all classical formulas of Φ, i.e., those formulas that do not involve the obligation operator. It is easy to prove that Φ_{CPL} is a CPL-theory and, clearly, $A \subseteq \Phi_{CPL}$. Since Φ is closed under the rules of $\mathcal{P}_3$, in particular those of the form $\{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\}$, we can conclude that $G(\Phi_{CPL}) \subseteq \Phi_{CPL}$. Then, we have that $G(\Phi_{CPL}) \subseteq \{\varphi : \mathbf{O}(\varphi) \in \Phi\}$ because Φ is closed under the rules of $\mathcal{P}_3$, in particular those of the form $\{\mathbf{O}(\psi) \leftarrow \varphi : \langle\varphi, \psi\rangle \in G\}$. Then, applying Cn to both, we can conclude that $Cn(G(\Phi_{CPL})) \subseteq Cn(\{\varphi : \mathbf{O}(\varphi) \in \Phi\})$. But, since

Φ is a SDL-theory, it is easy to prove that $\{\varphi : \mathbf{O}(\varphi) \in \Phi\}$ is a CPL-theory, and, therefore, $Cn(\{\varphi : \mathbf{O}(\varphi) \in \Phi\}) = \{\varphi : \mathbf{O}(\varphi) \in \Phi\}$. Let us summarize what we have prove up to now. We proved that, for every model Φ of $\mathcal{P}_3$ we have that, Φ_{CPL} is a CPL-theory such that $A \subseteq \Phi_{CPL}$ and $G(\Phi_{CPL}) \subseteq \Phi_{CPL}$. Moreover, we have that $Cn(G(\Phi_{CPL})) \subseteq \{\varphi : \mathbf{O}(\varphi) \in \Phi\}$. Using these conclusions together with the fact that $S_{\mathcal{P}_3} = \bigcap_{\Psi \in Mod(\mathcal{P}_3)} \Psi$, we can conclude that $\bigcap_{\Psi \in Mod(\mathcal{P}_3)} Cn(G(\Psi_{CPL})) \subseteq \bigcap_{\Psi \in Mod(\mathcal{P}_3)} \{\varphi : \mathbf{O}(\varphi) \in \Psi\} = \{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\}$. The result then follows from the observation that $out_3(G, A) \subseteq \bigcap_{\Psi \in Mod(\mathcal{P}_3)} Cn(G(\Psi_{CPL}))$. This last observation follows directly from the definition of $out_3(G, A)$ and the fact that $\{\Phi_{SDL} : \Phi \in Mod(\mathcal{P}_3)\} \subseteq \{B : A \subseteq B = Cn(B) \supseteq G(B)\}$.

We now prove the reverse inclusion, i.e., $\{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\} \subseteq out_3(G, A)$. Let $\mathcal{M}$ be the set of all CPL-theories T such that $A \subseteq T$ and $G(T) \subseteq T$. Consider the SDL-theory $\Phi_T = (\{\mathbf{O}(\psi) : \psi \in Cn(G(T))\} \cup T)^{\vdash_{SDL}}$ obtained from $T \in \mathcal{M}$. Using SDL reasoning, it is not hard to see that $(\Phi_T)_{CPL}$ is a CPL-theory. Also, we can check that Φ_T is a model of $\mathcal{P}_3$. Therefore, we have $S_{\mathcal{P}_3} \subseteq \Phi_T$ because $S_{\mathcal{P}_3}$ is the least model of $\mathcal{P}_3$. Then, we have that $\{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\} \subseteq \{\varphi : \mathbf{O}(\varphi) \in \Phi_T\}$. It is easy to see that $\{\varphi : \mathbf{O}(\varphi) \in \Phi_T\} = Cn(G(T))$. Therefore, we have $\{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\} \subseteq \bigcap_{T \in \mathcal{M}} \{\varphi : \mathbf{O}(\varphi) \in \Phi_T\} = \bigcap_{T \in \mathcal{M}} Cn(G(T)) = out_3(G, A)$. $\square$

Note that for the embedding in the lemma, standard normal logic programs are not enough. Not only those do not consider obligations but, equally important, those do not allow for complex formulas in the heads and bodies of rules. Bare in mind that the φ and ψ can be any classical propositional formulas. However, in this lemma we only needed to consider definite deontic logic programs, i.e. deontic logic programs without default negation. This is a consequence of the monotonicity of unconstrained IO logic. Contrarily, constrained IO logic has an intrinsic non-monotonic flavor and, as we will see below, default negation plays a fundamental role in the embedding.

In [10] it was showed that, given a generating set G, a set A of input formulas and assuming that we take out_3^+ as the out operation, there is a relation between the elements of $outfamily(G, A)$ and the default extensions of a Reiter's default system obtained from G and A. In fact, the default extensions are usually a strict subset of $outfamily(G, A)$, corresponding to the maximal elements. The following theorem can be seen as a strengthening of that result, as we prove that, for out_1^+ and out_3^+, we can capture the entire $outfamily$ using our stable models semantics.

Theorem 1. *Let G be a generating set, A an input set of classical propositional formulas, and let φ and ψ stand for classical propositional formulas.*

1) Consider the deontic logic program over an extended language that contains a constant $\overline{\mathbf{O}(\psi)}$ for every classical propositional formula $\psi \in L_{CPL}$.

$$\begin{aligned}\mathcal{P}_1 = &\{\mathbf{O}(\psi) \leftarrow \varphi, not\ \overline{\mathbf{O}(\psi)} : \langle\varphi, \psi\rangle \in G\} \cup \\ &\{\varphi \leftarrow : \varphi \in A\} \cup \\ &\{\mathbf{O}(\varphi) \leftarrow : \varphi \in A\} \cup \\ &\{\overline{\mathbf{O}(\psi)} \leftarrow not\ \mathbf{O}(\psi) : \psi \in L_{CPL}\}\end{aligned}$$

Then, taking out_1^+ as the out operator, we have

$$outfamily(G, A) = \bigcup_{T \in SM(\mathcal{P}_1)} \{\varphi : \mathbf{O}(\varphi) \in T\}.$$

2) *Consider the deontic logic program over an extended language that contains a constant $\overline{b}$ for every classical propositional formula $b \in L_{CPL}$.*

$$\begin{aligned}\mathcal{P}_3 = \{&\psi \leftarrow \varphi, not\ \overline{\psi} : \langle\varphi, \psi\rangle \in G\} \cup \\ \{&\varphi \leftarrow : \varphi \in A\} \cup \\ \{&\overline{\psi} \leftarrow not\ \psi : \psi \in L_{CPL}\}\end{aligned}$$

Then, taking out_3^+ as the out operator, we have

$$outfamily(G, A) = SM(\mathcal{P}_3)_{|L_{CPL}}.$$

Proof. We just prove condition 2). The proof of 1) is simpler than 2) and can be easily adapted from the proof of 2).

We prove that $outfamily(G, A) = SM(\mathcal{P}_3)_{|L_{CPL}}$ by proving the two inclusions separately. First of all, we prove that $outfamily(G, A) \subseteq SM(\mathcal{P}_3)_{|L_{CPL}}$. Let $H \in maxfamily(G, A)$, i.e., $H \subseteq G$ maximal such that $out_3^+(H, A)$ is consistent. Note that if $\langle\varphi, \psi\rangle \in G \setminus H$ then clearly $\psi \notin out_3^+(H, A)$. Consider now the set $\Phi = out_3^+(H, A) \cup Cn(\{\overline{\psi} : \psi \notin out_3^+(H, A)\})$. We prove that Φ is a stable model of $\mathcal{P}_3$. The calculation of the G-L transformation of $\mathcal{P}_3$ modulo Φ gives the definite program $\frac{\mathcal{P}_3}{\Phi} = \{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in H$ and $\psi \in out_3^+(H, A)\} \cup \{\overline{\psi} \leftarrow : \psi \notin out_3^+(H, A)\} \cup \{\varphi \leftarrow : \varphi \in A\}$. It is easy to see that the minimal model of $\frac{\mathcal{P}_3}{\Phi}$ is precisely Φ and, therefore, Φ is a stable model of $\mathcal{P}_3$. We now prove that $out_3^+(H, A) = \Phi_{|L_{CPL}}$. First of all, since $\{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in H$ and $\psi \in out_3^+(H, A)\} \subseteq \{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in H\}$ we can immediately conclude that $\Phi_{|L_{CPL}} \subseteq out_3^+(H, A)$. To prove the converse inclusion just note that $\Phi_{|L_{CPL}}$ satisfies every rule $\{\psi \leftarrow \varphi : \langle\varphi, \psi\rangle \in H\}$. This is the case because if $\langle\varphi, \psi\rangle \in H$ and $\psi \leftarrow \varphi \notin \frac{\mathcal{P}_3}{\Phi}$ then $\varphi \notin out_3^+(H, A)$. Since $out_3^+(H, A)$ is the minimal theory containing A which satisfies the rules of H we can conclude that $out_3^+(H, A) \subseteq \Phi_{|L_{CPL}}$.

Let us now prove that $SM(\mathcal{P}_3)_{|L_{CPL}} \subseteq outfamily(G, A)$. Let T be a stable model of $\mathcal{P}_3$. Consider $H = \{\langle\varphi, \psi\rangle \in G : \varphi \notin T$ or ψ is consistent with $T\}$. We need to prove two things: (1) $T_{|CPL} = out_3^+(H, A)$ and (2) H is maximal such that $out_3^+(H, A)$ is a consistent.

First of all, it is not hard to see that T is also a stable model of $\mathcal{P}_H$, where $\mathcal{P}_H$ is the program obtained from H just as $\mathcal{P}_3$ was obtained from G. To see this, note that, by definition, if $\langle\varphi, \psi\rangle \in G \setminus H$ then $\varphi \in T$ and ψ is inconsistent with T. This means that $\psi \notin T$ and, therefore, $\psi \leftarrow \varphi \notin \frac{\mathcal{P}_H}{T}$.

We now prove (1), i.e., that $T_{|CPL} = out_3^+(H, A)$. Recall that T is the minimal theory which is closed under the rules of $\frac{\mathcal{P}_H}{T}$. Note that T contains A because $\{\varphi \leftarrow : \varphi \in A\} \in \frac{\mathcal{P}_H}{T}$. Recall also that $out_3^+(H, A)$ is the minimal theory that contains A and it is closed under the rules of H. To conclude that $T_{|CPL} = out_3^+(H, A)$ we need to compare the rules of $\frac{\mathcal{P}_H}{T}$ with those of H. It is easy to see that if $\psi \leftarrow \varphi \in \frac{\mathcal{P}_H}{T}$ then $\langle \varphi, \psi \rangle \in H$. In fact, we just need to note that if $\psi \leftarrow \varphi \in \frac{\mathcal{P}_H}{T}$ then $\overline{\psi} \notin T$, which implies that $\psi \in T$. In that case, ψ is consistent with T and, therefore, we have that $\langle \varphi, \psi \rangle \in H$. So, we can immediately conclude that $T_{|CPL} \subseteq out_3^+(H, A)$. We now prove the reverse inclusion. It is not hard to see that $\langle \varphi, \psi \rangle \in H$ and $\psi \leftarrow \varphi \notin \frac{\mathcal{P}_H}{T}$ only if $\varphi \notin T$. Therefore, we can conclude that T is a theory which contains A and it is closed under H. Since $out_3^+(H, A)$ is the minimal one, we can conclude that $out_3^+(G, A) \subseteq T_{|CPL}$. Therefore, we can conclude that $T_{|CPL} = out_3^+(H, A)$.

We now prove (2), i.e., H is maximal such that $out_3^+(H, A)$ is consistent. Let $H' \subseteq G$ such that $H \subset H'$. Then, by definition of H, we have that $\langle \varphi, \psi \rangle \in H' \setminus H$ if $\varphi \in T$ and ψ is inconsistent with T. Using (1) we have that $T_{|CPL} = out_3^+(H, A) \subseteq out_3^+(H', A)$. Since $\varphi \in T$ we conclude that $\varphi \in out_3^+(H', A)$. Therefore, we have that $\psi \in out_3^+(H', A)$ because $\langle \varphi, \psi \rangle \in H'$. But since ψ is inconsistent with T it also inconsistent with $out_3^+(H', A)$. Therefore, $out_3^+(H', A)$ is itself inconsistent. □

One could wonder why, in the above theorem, and in the case of out_3^+, we did not need to consider deontic logic programs with deontic operators. The reason is that if we admit throughput, i.e., inputs to be part of the output, and reusability, i.e., outputs can be reused as inputs, the deontic reading of a pair $\langle \varphi, \psi \rangle$ is no longer accurate. This fact, which was already noticed in [12], happens because the reuse of outputs as inputs dilutes the difference between facts and the obligation of these facts. However, note that even though in the case of out_3^+ obligations are not used, standard logic programming is not enough for the above embedding. The reason is that, as already noted above after Lemma 1, we need to consider complex propositional formulas in the body and head of rules, something that is not possible in standard logic programs.

The above embedding theorem shows how we can recast IO logic in deontic logic programming. An interesting question now is what additional features can deontic logic programming immediately bring to IO logic.

First of all, it is very clear that deontic logic programs have a richer language. Note that the deontic logic programs necessary to embed IO logic are not very expressive compared with the expressivity of a normal deontic logic program. Moreover, in deontic logic programming we have an explicit use of default negation, which is fundamental to model exceptions. Also, deontic logic programs can have complex deontic formulas not only in the head, but also in the body of a rule. This is fundamental to model violations of obligations and to specify sanctions in case of violations.

Another fundamental notion that comes for free in the context of deontic logic programming is the notion of equivalence between normative systems. In [6] the notion of strong equivalence between deontic logic programs is presented. This notion is very suited to normative systems because if two normative systems are strongly equivalent (seen as deontic logic programs), then one can change one by the other in the middle of a larger normative system without changing the meaning of this system. More importantly, in [6] an extension of the so-called equilibrium logic is defined, which allows to check strong equivalence of deontic logic programs using logical equivalence.

We end this section with an example of the use of deontic logic programs, contrasted with input-output logic, in a contrary-to-duty situation.

Example 1. Contrary-to-duty paradoxes are very important in the area of deontic reasoning. Not only were they crucial for revealing some of the weaknesses of SDL in modeling norms, but, more importantly, they provided fundamental intuitions for the extensions of SDL that overcame some of these weaknesses. In a nutshell, contrary-to-duty paradoxes encode the problem of what obligations should follow from a normative system in a situation where some of the existing obligations are already being violated.

Consider the following contrary-to-duty paradox adapted from [14].

You should have neither a fence nor a dog. But, if you have a dog you should have both a fence and a warning sign. In a situation where you have a dog what obligations should hold?

As a first attempt to represent the statement, we can try using a direct reading

$$\mathbf{O}(\neg dog \wedge \neg fence) \leftarrow$$
$$\mathbf{O}(fence \wedge warningSign) \leftarrow dog$$

The problem is that, intuitively, this normative system is inconsistent. In fact, if dog is the case, then the conflicting obligations $\mathbf{O}(\neg dog \wedge \neg fence)$ and $\mathbf{O}(fence \wedge warningSign)$ both follow from the normative system. This reading is in accordance with, for example, Prakken and Sergot [14].

If we take a closer look at the description of the problem we can see that the first rule of the normative system wrongly does not distinguish between the two obligations appearing there. While the obligation not to have a dog is unconditional, the obligation not to have a fence is not. It has an exception: the case where you have a dog. Therefore, using deontic logic programs we can have a proper representation with the use of default negation to model this exception.

$$\mathbf{O}(\neg dog) \leftarrow \qquad\qquad \mathbf{O}(fence \wedge warningSign) \leftarrow dog$$
$$\mathbf{O}(\neg fence) \leftarrow not\ dog$$

Intuitively, the above normative system is no longer inconsistent. The rules for $\mathbf{O}(\neg fence)$ and $\mathbf{O}(fence \wedge warningSign)$ now have bodies that cannot hold at the same time (dog and $not\ dog$). Moreover, if we assume that we have both a dog and a $fence$ the consequences of the normative systems include $\{dog, fence, \mathbf{O}(\neg dog), \mathbf{O}(fence), \mathbf{O}(warningSign)\}$. Therefore, on the one hand, we are able to detect a violation of the obligation not to have a dog, and, on the other hand, the fact that we have a fence is not a violation, because the fact that there is a dog prevents the derivation of the obligation not to have a fence.

The following is the representation proposed in [12] of the cottage contrary-to-duty situation using IO logic.

$$\langle \mathbf{t},\ \neg(dog \vee fence)\rangle \qquad \langle dog,\ \ fence \wedge warningSign\rangle$$

The formula **t** stands for a tautology. As in our first attempt to model this situation using deontic logic programs, in IO logic the unconstrained output gives an excessive output whenever dog is the case. In fact, the output is not only inconsistent with the input, but it is also itself inconsistent. The output is itself inconsistent because it includes both $fence$ and $\neg fence$, and it is inconsistent with the input because it include $\neg dog$.

The use of constraint output solves this particular problem. In fact, we have that $maxfamily(G, A) = \{\{\langle dog,\ fence \wedge warningSign\rangle\}\}$ and $outfamily(G, A) = \{Cn(fence \wedge warningSign)\}$. Intuitively, since dog is the case, the conditional norm $\langle \mathbf{t},\ \neg(dog \vee fence)\rangle$ is always discarded and only the consequences of the other conditional norm are considered. Although in this particular formulation of the example the strategy behind the definitions of $maxfamily$ and $outfamily$ gives a reasonable solution, this is not always the case. As it was pointed out in [12], this strategy is very sensible to how the generating set is written, and in some case it cuts too deeply the output. As an example, suppose that we only consider the first conditional norm $\langle \mathbf{t},\ \neg(dog \vee fence)\rangle$. If dog is the case then, surprisingly, $outfamily(G, A)$ only contains the set of tautologies and therefore does not include $\neg fence$.

The motivation behind the idea of constraining the output to deal with contrary-to-duty situations is, as argued in [11], the fact that we should not consider obligations that are already being violated. In the cottage example, we should not conclude that it is obligatory not to have a dog because having a dog is seen as an unchangeable fact. Perhaps this argument is acceptable in the context of IO logic. We argue, however, that this kind of reasoning is not accurate if we want to reason about violation of obligations. In deontic logic programming we want (and can!) reason about the violation of obligations. Consider that, in the cottage example we have rules for applying sanctions in case of violations, i.e., we augment the above normative system with the rules $\mathbf{O}(fineD) \leftarrow \mathbf{O}(\neg dog), dog$ and $\mathbf{O}(fineF) \leftarrow \mathbf{O}(\neg fence), fence$. Then, given that we have a dog and a fence, the obligation $\mathbf{O}(fineD)$ is entailed by the system but $\mathbf{O}(fineF)$ is not. This kind of reasoning would not be possible we were assuming that $\mathbf{O}(\neg dog)$ should not follow from the normative system when dog is the case.

6 Conclusions

We started this paper by introducing a framework for representing and reasoning about normative systems – deontic logic programs – which combines the expressivity of standard deontic logic with non-monotonic logic programs.

We have shown how deontic logic programs may embed, in a natural way, the original input-output logic of [10]. Moreover, making use of the non-monotonic features of logic programming, we were also able to embed in deontic logic programs the extension of input-output logic of [11], that is able to cope with contrary-to-duty paradoxes. This latter result can be seen as a strengthening of the existing weak connection drawn in [10] between input-output logic and Reiter's default logic.

We then contrasted the use of deontic logic programs with that of IO logic in an example of a contrary-to-duty situation. In IO logic, these situations may be handled by the use of constraint output. However, as it was pointed out in [12], this strategy is very sensible to how the generating set is written, and in some case it cuts too deeply the output. This is not the case in deontic logic programs and, further, one can reason about violation of obligations. Moreover, with deontic logic programs one can more easily model exceptions, by the explicit use of default negation; and also model violations of obligations and specify sanctions in case of violations, by being able to deal with complex deontic formulas also in the body of rules. Guided by the recasting of IO logic in deontic logic programs, an interesting topic for future work is how to extend IO logic to incorporate these important features.

Another important possibility for future work, opened by the results in this paper, is the study of equivalence, and strong equivalence, between normative systems in IO logic. We think that strong equivalence is very suited notion in normative systems: if two normative systems are strongly equivalent, then one can change one by the other in the middle of a larger normative system without changing the meaning of this system, thus opening the way to (modular) simplifications of normative systems.

References

1. J. Carmo and A. Jones. Deontic logics and contrary-to-duties. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, vol. 8*, pages 265–343, 2002.
2. B. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.
3. R. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24(2):33–36, 1963.
4. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP*, pages 1070–1080. MIT Press, 1988.
5. R. Gonçalves and J. J. Alferes. Parametrized logic programming. In T. Janhunen and I. Niemelä, editors, *Logics in Artificial Intelligence – JELIA*, volume 6341 of *LNCS*, pages 182–194. Springer, 2010.
6. R. Gonçalves and J. J. Alferes. Parametrized equilibrium logic. In J. P. Delgrande and W. Faber, editors, *LPNMR*, volume 6645 of *LNCS*, pages 236–241. Springer, 2011.
7. G. Governatori and A. Rotolo. Bio logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17(1):36–69, 2008.
8. J. F. Horty. Deontic logic as founded on nonmonotonic logic. *Ann. Math. Artif. Intell.*, 9(1-2):69–91, 1993.
9. D. Lewis. *Semantic analyses for dyadic deontic logic*. Cambridge University Press, 1999.
10. D. Makinson and L. van der Torre. Input-output logics. *Journal of Philosophical Logic*, 29:383–408, 2000.
11. D. Makinson and L. van der Torre. Constraints for input/output logics. *Journal of Philosophical Logic*, 30:155–185, 2001.
12. D. Makinson and L. van der Torre. What is input/output logic? input/output logic, constraints, permissions. In G. Boella, L. van der Torre, and H. Verhagen, editors, *Normative Multi-agent Systems*, volume 07122 of *Dagstuhl Seminar Proceedings*, 2007.
13. B. Motik and R. Rosati. Reconciling description logics and rules. *J. ACM*, 57(5), 2010.
14. H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.
15. L. van der Torre. Contextual deontic logic: Normative agents, violations and independence. *Ann. Math. Artif. Intell.*, 37(1-2):33–63, 2003.
16. G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.
17. R. Wójcicki. *Theory of Logical Calculi*. Synthese Library. Kluwer Academic Publishers, 1988.