Journal of Applied Non-Classical Logics, 2013
Vol. 23, Nos. 1–2, 25–48, http://dx.doi.org/10.1080/11663081.2013.798996

Equivalence of defeasible normative systems

José Júlio Alferes, Ricardo Gonçalves* and João Leite

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

Normative systems have been advocated as an effective tool to regulate interaction in multi-agent systems. The use of deontic operators and the ability to represent defeasible information are known to be two fundamental ingredients to represent and reason about normative systems. In this paper, after introducing a framework that combines standard deontic logic and non-monotonic logic programming, deontic logic programs (DLP), we tackle the fundamental problem of equivalence between normative systems using a deontic extension of David Pearce's Equilibrium Logic and its monotonic basis, the logic of Here-and-There. We also show how deontic logic programs can be used to represent and reason about normative systems, and establish a strong connection with input-output logic.

Keywords: logic programming; deontic logic; equilibrium logic

1. Introduction

Normative systems are effective tools for regulating interaction in multi-agent systems. In such systems, norms encode desirable behaviours for a population of a natural or artificial society. In general, they are commonly understood as rules specifying what is expected to follow (obligations, permissions, etc.) from a specific set of facts. In order to encourage agents to act according to the norms, normative systems should also be able to specify the application of rewards/sanctions. With respect to the semantics, such normative frameworks should have a simple and declarative semantics. This would allow the agents (the ones that are subject to the normative system), the modeller (the one that writes down the norms) and the electronic institution (the one responsible for monitoring the agents and applying the sanctions/rewards) to reason about the normative system in a simple and clear way.

A deontic logic is a logic that deals with the notions of obligation, permission and prohibition, and it is, therefore, a fundamental tool for modelling reasoning in normative systems. Since the seminal work of von Wright (1951), many have investigated and developed systems of deontic logic (Hilpinen, 1981). One such system is the modal logic KD, usually known as Standard Deontic Logic (SDL; Chellas, 1980). Although accepted as a fundamental tool for modelling normative assertions, SDL has proven insufficient for the task of representing norms (Chisholm, 1963). For instance, its inability to deal with some paradoxes such as those involving the so-called contrary-to-duty obligations is well known. In a nutshell, contrary-to-duty paradoxes encode the problem of determining what obligations follow from a normative system in a situation where some of the existing obligations are already being violated. Contrary-to-duty paradoxes are very

*Corresponding author. Email: rjrg@fct.unl.pt important in the area of deontic reasoning, not only because they revealed the weaknesses of SDL in modelling norms, but, more importantly, because they provided fundamental intuitions for the extensions of SDL that overcame some of these weaknesses.

Roughly, we can say that the main difficulty of SDL is the fact that classical implication does not provide a faithful representation of the conditional obligations that usually populate a normative system. Consider the following contrary-to-duty normative statement taken from Prakken and Sergot (1996, p. 91), which presents two simple norms about a holiday cottage regulation:

> You should not have a fence. But, if you have one, it should be white. In a situation where you have a fence what obligations should hold?

This normative statement could easily be represented in SDL using the formulas $\mathbf{O}(\neg fence)$ and $fence \Rightarrow \mathbf{O}(whiteFence)$. The problem is that, if we consider that $fence$ and the reasonable formula $whiteFence \Rightarrow fence$ are the case, then this natural representation of the normative statement is inconsistent in SDL.

Several approaches modelling conditional obligations have been proposed and shown to have a more reasonable behaviour in the face of the aforementioned paradoxes. These approaches include, for example, works using dyadic modal logics (Lewis, 1999; Prakken & Sergot, 1996; van der Torre, 2003), where a dyadic deontic operator $\mathbf{O}_A B$ is introduced with the reading that B is obligatory in the sub-ideal situation where A is the case, and input-output logic (Makinson & van der Torre, 2001) in which conditional norms are represented as pairs $\langle A, B\rangle$ with the reading that if A is the case then B is obligatory. All of these approaches encode conditional obligations using a rule-based framework.

Another fundamental ingredient for a language capable of modelling norms is the ability to express defeasible knowledge. This is important, for example, to represent exceptions, which are very common in normative rules.

Several approaches using non-monotonic logics were applied to the problem of representing and reasoning about norms. Besides being able to model exceptions, another important feature of the approaches for defeasible knowledge is that they are usually rule-based frameworks. This allows for the modelling of the rule-based flavour of norms. Important works on this topic include, for example, the seminal work in Sergot et al. (1986), in which the British Nationality Act (1981) is represented using logic programming; the work in Ryu and Lee (1993), which uses a clausal form of logic programs where the deontic concepts are captured by first-order representations; the work in Brewka (1996), which uses logic programs with dynamic preferences under the well-founded semantics; the work in Prakken and Sartor (1997), which represents normative systems using logic programs with defeasible priorities; the work in Boella, Governatori, Rotolo, and van der Torre (2010), which uses defeasible logic for representing normative systems; and the recent work of Alberti, Gomes, Gonçalves, Leite, and Slota (2011), and Alberti et al. (2012), who use an expressive hybrid combination of logic programs and description logics.

The representation and reasoning about normative systems would greatly benefit from a framework combining deontic logic and rule-based non-monotonic reasoning. Unsurprisingly, some attention has been devoted to the combination of these two notions (Nute, 1997). Important approaches include the work in McCarty (1994), which presents a combination of dyadic deontic logic with a default reasoning system, with explicit exceptions, based on intuitionistic logic; the work of Governatori & Rotolo (2008) using deontic rules in defeasible logic; and the work of Horty (1993), which presents a non-monotonic consequence relation for dyadic deontic logic, tailored for reasoning about conflicting obligations and conditional obligations. There is also the non-monotonic version of input-output logic (Makinson & van der Torre, 2000), tailored to reason about contrary-to-duty situations.

We can, nevertheless, point out several limitations to the above combinations of deontic logic and non-monotonic reasoning. The work in Horty (1993) has a quite limited language, only allowing reasoning about sets of dyadic obligations. In Makinson and van der Torre (2000) deontic formulas are not allowed to appear as conditions in norms. In Governatori and Rotolo (2008), and McCarty (1994), complex formulas are not allowed to appear in the scope of a deontic operator. Moreover, the language of Governatori and Rotolo (2008) does not have a declarative semantics, but rather a proof-theoretical semantics. Finally, most of the above approaches do not allow or have a highly limited use of explicit exceptions.

None of the above approaches, therefore, succeed in complying with all of the following syntactical and semantical requirements: from a syntactical point of view, a framework for specifying and reasoning with normative systems should have a rich language, thus allowing complex deontic formulas to appear in both the body and the head of rules. With respect to the semantics, such normative frameworks should have a declarative semantics that is easy to understand.

In this paper we propose a language for representing and reasoning about normative systems that combines deontic logic with logic programming with default negation. We give it a fully declarative semantics, allowing the definition of a strong notion of equivalence between normative systems.

Two major features distinguish our approach from other formalisms that combine deontic operators with non-monotonic reasoning. First of all, the richness of our language allows complex deontic logic formulas to appear in both the body and the head of a rule, combined with the use of default negation. Additionally, we endow the normative systems with a purely declarative semantics, which stems from the stable model semantics of logic programs. Furthermore, by making use of Pearce's (2006) results on Equilibrium Logic and its relation to strong equivalence in logic programs (Lifschitz, Pearce, & Valverde, 2000), we can define the fundamental notion of equivalence between normative systems, and, more importantly, develop a logic which allows us to verify equivalence of normative systems by using logical equivalence.

We end the paper by showing that our framework is general enough to embed input-output logic (Makinson & van der Torre, 2000), and has advantages over this logic in that it can easily be used to represent and reason about violation of obligations. Interestingly, the embedding makes use of both non-stratified negation and of disjunction in logic programs, showing that the generality of the stable models semantics is indeed needed.

The paper is structured as follows: in Section 2 we introduce the framework for normative systems. Then, in Section 3, we define the semantics of normative systems. In Section 4 we present some examples to help us clarify the syntax and semantics of our language. The study of the notion of equivalence in normative systems is undertaken in Section 5. In Section 6 we prove an embedding result for input-output logic. Finally, in Section 7, we draw some conclusions and point out paths for future research.

2. Framework

In this section we introduce the main ingredients for setting up a normative framework that jointly combines the expressivity of deontic logic with that of non-monotonic rules from logic programming. First though, we briefly recap standard deontic logic.

2.1. Standard deontic logic

The formal study of deontic logic was highly influenced by modal logic. In fact, standard deontic logic (SDL) is a modal logic with two modal operators, one for obligation and another for permission. We briefly describe SDL, and refer the reader to Chellas (1980) for further details.

Formally, the language of SDL, dubbed L_{SDL}, is constructed from a set $Prop$ of propositional symbols using the usual classical connectives $\sim, \Rightarrow$, and the unary deontic operator $\mathbf{O}$ (obligation). The classical connectives $\vee$ and $\wedge$ are defined as abbreviations in the usual way. Moreover, the permission and forbidden operators are defined as $\mathbf{P} :=\sim \mathbf{O} \sim$ and $\mathbf{F} := \mathbf{O} \sim$ respectively.

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 in that world. Moreover, the relation R is assumed to be serial, i.e., for every $w \in W$ there exists $w' \in W$ such that wRw'. As usual, we define the satisfaction of a formula $\varphi \in L_{\text{SDL}}$ in a Kripke model $\mathcal{M} = \langle W, R, \mathcal{V}\rangle$ at a world $w \in W$, by induction on the structure of φ:

(1) $\mathcal{M}, w \Vdash p$ if $p \in \mathcal{V}(w)$, for $p \in Prop$;
(2) $\mathcal{M}, w \Vdash \sim \varphi$ if $\mathcal{M}, w \nVdash \varphi$;
(3) $\mathcal{M}, w \Vdash \varphi_1 \Rightarrow \varphi_2$ if $\mathcal{M}, w \nVdash \varphi_1$ or $\mathcal{M}, w \Vdash \varphi_2$;
(4) $\mathcal{M}, w \Vdash \mathbf{O}(\varphi)$ if $\mathcal{M}, w' \Vdash \varphi$ for every w' such that $\langle w, w'\rangle \in R$.

We say that an SDL formula φ is a *logical consequence* of a set Φ of SDL formulas, written $\Phi \vdash_{\text{SDL}} \varphi$, 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 an SDL *theorem* if $\emptyset \vdash_{\text{SDL}} \varphi$.

At this point it is important to stress that the consequence relation defined above is the so-called local consequence relation, which can be 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 $\delta \in \Phi$. The local and the global consequence relations are quite different, and the reason why this difference is sometimes neglected is because the interest is only in the set of theorems, which is the same for both consequences. From our point of view, the global consequence does not faithfully represent normative reasoning. This can easily be seen using the fact that $\varphi \vdash_g \mathbf{O}(\varphi)$, which is not valid reasoning if $\mathbf{O}$ is to be read as an obligation. Therefore, since, as we shall see, the notion of consequence relation is a fundamental tool in our framework, we do not neglect the difference between the two consequences and work just with the local one, which is more adequate for normative reasoning.

Let us now continue by defining the notion of logical theory. As we will see, logical theories play a fundamental role in the definition of the semantics for normative systems.

Definition 1. *A set of SDL formulas Φ is said to be an SDL logical theory if Φ is closed under SDL consequence, i.e., for every $\varphi \in L_{\text{SDL}}$ if $\Phi \vdash_{\text{SDL}} \varphi$ then $\varphi \in \Phi$.*

We denote by Th_{SDL} the set of theories of SDL. A fundamental property of the set of theories is that the tuple $\langle Th_{\text{SDL}}, \subseteq\rangle$ is a complete lattice with the smallest element the set $Theo_{\text{SDL}}$ of theorems of SDL and the greatest element the set L_{SDL} of all SDL formulas. One of the consequences of the fact that $\langle Th_{\text{SDL}}, \subseteq\rangle$ is a complete lattice is that, given any subset A of L_{SDL}, there exists the smallest SDL theory that contains A, which we denote by $A^{\vdash_{\text{SDL}}}$. The theory $A^{\vdash_{\text{SDL}}}$ is also called the theory generated by A.

2.2. Deontic logic programs

Deontic logic programs are composed of rules that resemble the usual disjunctive logic program rules, but where complex SDL formulas can appear where usually only atoms are allowed.

Definition 2. *A deontic logic program is a set of rules*

$$\varphi_1; \dots; \varphi_k \leftarrow \psi_1, \dots, \psi_n, not\ \delta_1, \dots, not\ \delta_m \tag{1}$$

where $\varphi_1, \dots, \varphi_k, \psi_1, \dots, \psi_n, \delta_1, \dots, \delta_m \in L_{SDL}$.

As usual, the symbol $\leftarrow$ represents rule implication, the symbol ',' represents conjunction, the symbol ';' represents disjunction and the symbol *not* represents default negation. A rule of the form (1) has the usual reading that whenever $\psi_1, \dots, \psi_n$ hold and $\delta_1, \dots, \delta_m$ are not known to hold, then at least one φ_i should hold.

A deontic logic program is said to be *non-disjunctive* if, for each rule in the program, we have that $k = 1$, i.e., each rule has exactly one formula in its head. A *definite deontic logic program* is a set of rules without default negation, i.e., of the form $\varphi_1; \dots; \varphi_k \leftarrow \psi_1, \dots, \psi_n$. We should note that non-disjunctive deontic logic programs can be seen as a particular case of the general construction presented in Gonçalves and Alferes (2010).

Contrarily to some works in the literature (Governatori & Rotolo, 2008; Horty, 1993; Makinson & van der Torre, 2000), our framework admits deontic formulas in both the body and the head of rules, and such formulas can be complex and not just atomic. Additionally, we can make explicit use of default negation. This extra flexibility is relevant, for example, to dealing with non-compliance and application of sanctions. For example, we can use the rule

$$\mathbf{O}(payFine \vee (apologise \wedge paySameAmount)) \leftarrow \mathbf{O}(pay), not\ pay$$

to express that if an agent has the obligation to pay some bill, and it is not known that the agent has paid it, then the agent is obliged to pay a fine or apologise and pay the same amount.

2.3. Normative systems

We now define the central concept of normative systems. A normative system is usually understood as a set of rules that specify what obligations and permissions follow from a given set of facts, and, additionally, specify which sanction/rewards should apply. In our approach, we directly use the deontic logic programs introduced in the previous section to represent a normative system.

Definition 3. *A normative system* $\mathcal{N}$ *is a deontic logic program.*

Another important notion is that of a set of facts.[1] A set of facts not only represents the state of the environment that is populated by the agents (using formulas without deontic operators), but also the normative state (using formulas with deontic operators).

Definition 4. *A set of facts* $\mathcal{F}$ *is a set of SDL formulas.*

A set of facts $\mathcal{F}$ is said to be *consistent* if it is a consistent set of SDL formulas, i.e., there exists $\varphi \in L_{\text{SDL}}$ such that $\mathcal{F} \nvdash_{\text{SDL}} \varphi$. In what follows, we abuse notation and often write $\mathcal{F}$ to refer to its correspondent set of deontic logic program's rules $\{\varphi \leftarrow : \varphi \in \mathcal{F}\}$.

In normative multi-agent systems (Tinnemeier, Dastani, & Meyer, 2009) there is usually the distinction between the so-called *brute facts*, which represent the state of the environment shared by the agents, and the *institutional facts*, which represent the normative/institutional state of the multi-agent system. In this paper we adopt a simplified notion of set of facts, by not distinguishing between brute and institutional facts. Nevertheless, the richness of our language allows us to incorporate this distinction easily. We just need to consider formulas without deontic operators as brute facts and formulas with deontic operators as institutional facts.

3. Semantics

In order to allow agents and institutions to reason about a normative system, it is very important that it has a rigorous formal semantics which, at the same time, should be clean and as simple as possible.

In this section we present a declarative semantics for normative systems, by defining a stable model-like semantics (Gelfond & Lifschitz, 1988) for deontic logic programs.

3.1. Stable model semantics

Given the richness of the language of deontic logic programs, which allows complex SDL formulas in the head and body of its rules, instead of only atoms, its semantics is not a straightforward exercise. The problem resides in the fact that, contrarily to atoms, these formulas are not independent. A key idea for overcoming this difficulty is to define a notion of interpretation that accounts for such interdependence between these '*complex atoms*'.

Definition 5. *An interpretation T is a theory of SDL.*

Recall that a theory of SDL is a set of SDL formulas that is closed under SDL logical consequence. The key idea of taking theories of SDL as interpretations contrasts with the usual definition of an interpretation as any set of atoms, and allows the semantics to cope with the interdependence between the SDL formulas appearing in the rules.

Definition 6. *An interpretation T satisfies a rule*

$$\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n, \mathit{not}\ \delta_1, \ldots, \mathit{not}\ \delta_m$$

if, whenever $\psi_i \in T$ for every $i \in \{1, \ldots, n\}$ and $\delta_j \notin T$ for every $j \in \{1, \ldots, m\}$, we have that $\varphi_l \in T$ for some $l \in \{1, \ldots, k\}$.

An interpretation is a *model* of a deontic logic program $\mathcal{P}$ if it satisfies every rule of $\mathcal{P}$. We denote by $Mod(\mathcal{P})$ the set of models of $\mathcal{P}$. We take set theoretical inclusion as the ordering over interpretations. An interpretation T is said to be a *minimal model* of a program $\mathcal{P}$ if T is a model of $\mathcal{P}$ and there exists no model T' of $\mathcal{P}$ such that $T' \subset T$.

To assign semantics to deontic logic programs we follow the standard path of first assigning semantics to definite deontic logic programs, i.e., programs without default negation, and then use this semantics to define the semantics of deontic logic programs by first eliminating all occurrences of default negation.

Recall that in the case of a definite disjunctive logic program, the set of its stable models is the set of its minimal models.

In our framework we define the semantics of definite deontic logic programs in a similar way, i.e., the set of stable models of a definite deontic logic program is the set of its minimal models. It is easy to see that, similar to what happens with definite non-disjunctive logic programs, a definite non-disjunctive deontic logic program has a unique least model.

To define the stable model semantics of a deontic logic program we use a Gelfond–Lifschitz-like operator.

Definition 7. *Let $\mathcal{P}$ be a deontic logic program and T an interpretation. The GL-transformation of $\mathcal{P}$ modulo T is the definite deontic logic program $\frac{\mathcal{P}}{T}$ obtained by:*

- *removing from $\mathcal{P}$ all rules which contain a default not φ such that $\varphi \in T$;*
- *removing from the remaining rules of $\mathcal{P}$ all default negated formulas.*

Definition 8. *An interpretation T is a* stable model *of a deontic logic program $\mathcal{P}$ if T is a minimal model of $\frac{\mathcal{P}}{T}$. We denote by $SM(\mathcal{P})$ the set of all stable models of $\mathcal{P}$. An SDL formula φ is true under the stable model semantics of $\mathcal{P}$, denoted by $\mathcal{P} \vDash_{SM} \varphi$, if it belongs to every stable model of $\mathcal{P}$.*

We can use the semantics of deontic logic programs to define the semantics of normative systems given a particular set of facts.

Definition 9. *Let $\mathcal{N}$ be normative system and $\mathcal{F}$ a set of facts. Then, an SDL formula φ is said to be a consequence of $\mathcal{N}$ given $\mathcal{F}$ iff $\mathcal{N} \cup \mathcal{F} \vDash_{SM} \varphi$. We denote by $SM_{\mathcal{F}}(\mathcal{N})$ the set of all such consequences.*

4. Examples

In this section we present some examples to illustrate the flexibility of the syntax and semantics of deontic logic programs.

Example 10. *Consider the normative system composed of the following rules:*

$$\begin{aligned} \mathbf{O}(b \Rightarrow a) &\leftarrow not\ \mathbf{O}(c \wedge d) \\ \mathbf{O}(a \vee b) &\leftarrow \\ p &\leftarrow \mathbf{O}(a) \\ q &\leftarrow \mathbf{O}(c \wedge \sim a), p. \end{aligned}$$

This normative system is as interesting illustration of the use of deontic reasoning in the computation of the stable models of a normative system. If the set of facts is empty, the unique stable model of this normative system is $\{\mathbf{O}(b \Rightarrow a), \mathbf{O}(a \vee b), p\}^{\vdash_{SDL}}$. Note that this stable model contains every consequence of the set $\{\mathbf{O}(b \Rightarrow a), \mathbf{O}(a \vee b), p\}$. For example, it contains $\mathbf{O}(a)$. It is relevant to note that p being a consequence of the normative system follows from the fact that $\mathbf{O}(b \Rightarrow a)$ and $\mathbf{O}(a \vee b)$ together entail $\mathbf{O}(a)$ in SDL, i.e., $\{\mathbf{O}(b \Rightarrow a), \mathbf{O}(a \vee b)\} \vdash_{SDL} \mathbf{O}(a)$. Therefore, the third rule allows us to conclude p.

Example 11. *Consider the following normative systems $\mathcal{N}_1$ and $\mathcal{N}_2$:*

$$\begin{aligned} \mathbf{O}(a \vee b) &\leftarrow & \qquad \mathbf{O}(a); \mathbf{O}(b) &\leftarrow \\ p &\leftarrow \mathbf{O}(a) & p &\leftarrow \mathbf{O}(a) \\ p &\leftarrow \mathbf{O}(b) & p &\leftarrow \mathbf{O}(b). \end{aligned}$$

These normative systems are interesting because they present two types of disjunction that can be used in the head of a rule. In $\mathcal{N}_1$ we have the obligation of the classical disjunction of a and b. In $\mathcal{N}_2$ we have the logic programming disjunction of obligation of a and obligation of b. To help clarify the difference between them, let us present the semantics of $\mathcal{N}_1$ and $\mathcal{N}_2$. The unique stable model of $\mathcal{N}_1$ is the set $\{\mathbf{O}(a \vee b)\}^{\vdash_{SDL}}$. In this case, we have that $\mathbf{O}(a \vee b)$ is a consequence of $\mathcal{N}_1$. Nevertheless p is not a consequence of $\mathcal{N}_1$ because from $\mathbf{O}(a \vee b)$ we cannot conclude in SDL either $\mathbf{O}(a)$ or $\mathbf{O}(b)$. Note that even if the first rule of $\mathcal{N}_1$ is $\mathbf{O}(a) \vee \mathbf{O}(b) \leftarrow$, we do not have that p is a consequence of $\mathcal{N}_1$, since neither $\mathbf{O}(a)$ nor $\mathbf{O}(b)$ follow from $\mathbf{O}(a) \vee \mathbf{O}(b)$ in SDL.

In the case of the normative system $\mathcal{N}_2$, we have two stable models $\{\mathbf{O}(a), p\}$ and $\{\mathbf{O}(b), p\}$. The logic programming disjunction forces every model of $\mathcal{N}_2$ to have either $\mathbf{O}(a)$ or $\mathbf{O}(b)$. Therefore, using the second or the third rule we can always conclude p.

Example 12. *Consider the normative system composed of the following rules, which is an example of a non-stratified program (note the cyclic dependency between* $\mathbf{O}(a)$ *and* $\mathbf{O}(b)$ *in the second and third rules):*

$$\begin{aligned}\mathbf{O}(a \Rightarrow e) &\leftarrow \\ \mathbf{O}(a) &\leftarrow not\ \mathbf{O}(b) \\ \mathbf{O}(b) &\leftarrow not\ \mathbf{O}(a) \\ \mathbf{O}(c) &\leftarrow \mathbf{O}(e), \mathbf{O}(d) \\ \mathbf{O}(d) &\leftarrow not\ \mathbf{O}(c).\end{aligned}$$

First, note that a normative system with just the first three rules has two stable models, $\{\mathbf{O}(b), \mathbf{O}(a \Rightarrow e)\}^{\vdash_{SDL}}$ *and* $\{\mathbf{O}(a), \mathbf{O}(a \Rightarrow e)\}^{\vdash_{SDL}}$*. Also note that the fourth one also contains* $\mathbf{O}(e)$ *(since it is, as all interpretations, closed under SDL consequence).*

It is not difficult to see that the addition of the remaining rules disallows $\mathbf{O}(e)$ *from being part of a stable model of the normative system. Therefore, the unique stable model of this normative system is the set* $\{\mathbf{O}(b), \mathbf{O}(a \Rightarrow e), \mathbf{O}(d)\}^{\vdash_{SDL}}$*.*

Example 13. *Consider the normative system composed of the following rules*

$$\begin{aligned}p &\leftarrow \sim \mathbf{O}(\sim a) \\ q &\leftarrow not\ \mathbf{O}(\sim a).\end{aligned}$$

This normative system is interesting because it shows that our framework is expressive enough to capture two different notions of permission: explicit permission and permission by lack of prohibition.[2]

The unique stable model of this normative system is the set $\{q\}^{\vdash_{SDL}}$*. With it one can conclude q from the absence of an explicit prohibition of a (recall that, according to the abbreviations given in Section 2.1,* $\mathbf{O}(\sim a)$ *is the same as* $\mathbf{F}(a)$*, i.e., a is forbidden). However, one cannot conclude p, because, although a is not forbidden, it is also not explicitly permitted (recall that, again according to the abbreviations given in Section 2.1,* $\sim \mathbf{O}(\sim a)$ *is the same as* $\mathbf{P}(a)$*, i.e., a is permitted).*

If we add the explicit permission of a as a fact of the normative system, i.e., we add the rule $\sim \mathbf{O}(\sim a) \leftarrow$ *to the above normative system, then both p and q follow from the normative system.*

We end this section by revisiting the example from the Introduction.

Example 14. *Let us consider the following normative statement (cf. Makinson & van der Torre, 2007, p.5), which is an adaptation and extension of the cottage example:*

> 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?

The following is a first attempt to represent the above normative statement using our framework.

$$\begin{aligned}\mathbf{O}(\sim dog \wedge \sim fence) &\leftarrow \\ \mathbf{O}(fence \wedge warningSign) &\leftarrow dog.\end{aligned}$$

The problem is that, intuitively, there are circumstances in which this normative system can lead to inconsistency. In fact, if $\mathcal{F} = \{dog\}$*, the conflicting obligations* $\mathbf{O}(\sim dog \wedge \sim fence)$ *and*

$\mathbf{O}(fence \wedge warningSign)$ both follow from the normative system. This reading is in accordance with, for example, Prakken and Sergot (1996).

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 defeasible. It has an exception: the case where you have a dog. Therefore, a proper representation should use default negation to model this exception.

$$\begin{aligned} \mathbf{O}(\sim dog) &\leftarrow \\ \mathbf{O}(\sim fence) &\leftarrow not\ dog \\ \mathbf{O}(fence \wedge warningSign) &\leftarrow dog. \end{aligned}$$

Intuitively, in the above normative system, the rules for $\mathbf{O}(\sim fence)$ and $\mathbf{O}(fence \wedge warningSign)$ are no longer conflicting, since they now have bodies that cannot hold at the same time (dog and $not\ dog$).

Suppose that it is not known whether you have a dog, i.e., $\mathcal{F} = \{\}$. Then

$$SM_{\mathcal{F}}(\mathcal{N}) = \{\mathbf{O}(\sim dog), \mathbf{O}(\sim fence)\}^{\vdash_{SDL}},$$

i.e., the obligation $\mathbf{O}(\sim fence)$ follows from the normative system and $\mathbf{O}(warningSign)$ does not follow.

Suppose now that $\mathcal{F} = \{dog\}$ is the current set of facts. Then

$$SM_{\mathcal{F}}(\mathcal{N}) = \{dog, \mathbf{O}(\sim dog), \mathbf{O}(fence), \mathbf{O}(warningSign)\}^{\vdash_{SDL}}.$$

Let the set of facts be $\mathcal{F} = \{dog, fence\}$. The consequences of $\mathcal{N} \cup \mathcal{F}$ include

$$\{dog, fence, \mathbf{O}(\sim dog), \mathbf{O}(fence), \mathbf{O}(warningSign)\}.$$

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. We argue that this kind of reasoning is relevant. Consider, for example, that there are rules for applying sanctions in case of violations, i.e., we augment $\mathcal{N}$ with the rules

$$\begin{aligned} \mathbf{O}(fineD) &\leftarrow \mathbf{O}(\sim dog), dog \\ \mathbf{O}(fineF) &\leftarrow \mathbf{O}(\sim fence), fence. \end{aligned}$$

Then, given the set of facts $\mathcal{F} = \{dog, fence\}$, the obligation $\mathbf{O}(fineD)$ is entailed by the normative system but $\mathbf{O}(fineF)$ is not.

5. Equivalence of normative systems

In this section we study the fundamental problem of equivalence of normative systems. We start by introducing the notion of equivalence of normative systems using the equivalence of deontic logic programs. We then define a deontic extension of the logic of here-and-there (HT; Heyting, 1930), and prove that logical equivalence in this logic provides a necessary and sufficient condition for checking strong equivalence of normative systems.

5.1. Equivalence of deontic logic programs

The notion of equivalence is very important in the area of logic programming. Usually two logic programs are dubbed equivalent if they have the same stable model semantics. Nevertheless, it became evident that this definition of equivalence is too weak in the sense that it is not immune to the addition of contexts. For example, consider that, given a program $\mathcal{P}$, we want to replace part of $\mathcal{P}$ with an equivalent set of rules. There is no guarantee that the resulting program is equivalent to $\mathcal{P}$. Therefore, a stronger notion of equivalence between logic programs, dubbed strong equivalence, was defined precisely as equivalence under the presence of any context (Lifschitz et al., 2000).

We now generalise this notion of strong equivalence to deontic logic programs.

Definition 15. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two deontic logic programs. We say that $\mathcal{P}_1$ and $\mathcal{P}_2$ are strongly equivalent, denoted by $\mathcal{P}_1 \equiv_s \mathcal{P}_2$, if for any deontic logic program $\mathcal{P}$, the programs $\mathcal{P}_1 \cup \mathcal{P}$ and $\mathcal{P}_2 \cup \mathcal{P}$ have the same stable models.*

This notion of strong equivalence provides a robust definition of equivalence between normative systems in the sense that two equivalent normative systems have the same semantics even in the presence of an additional set of rules. This robustness can also be seen from a different, but equally important, perspective. Imagine that we have a normative system and we want to substitute part of it with an equivalent simplification. In this case, only the use of strong equivalence guarantees that the semantics of the entire normative system is not affected by this change.

As it is standard practice in other notions of equivalence of normative systems to check strong equivalence of two normative systems (e.g., Makinson & van der Torre, 2000), we need to check if the normative systems are equivalent in the presence of any additional set of norms. This is not usually possible because the number of possible contexts is infinite.

It would be desirable to have a mechanism to prove strong equivalence of two normative systems by just comparing them. In fact, contrary to other approaches, in Section 5.2, inspired by David Pearce's results on strong equivalence of logic programs, we define a logic in which we can check strong equivalence of deontic logic programs just by checking logical equivalence.

In some situations the notion of strong equivalence may be seen as too strong—for example, when there are restrictions on the kind of predicates describing the environment (i.e., those that can be added as facts). Interestingly, in logic programming there is a weaker notion of equivalence, called *uniform equivalence* (Eiter & Fink, 2003; Pearce & Valverde, 2004), which restricts the contexts that can be added to the programs. We generalise this notion of uniform equivalence to the framework of deontic logic programs.

Definition 16. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two deontic logic programs. We say that $\mathcal{P}_1$ and $\mathcal{P}_2$ are uniformly equivalent, denoted by $\mathcal{P}_1 \equiv_u \mathcal{P}_2$, if for any set $\mathcal{F}$ of deontic formulas, the programs $\mathcal{P}_1 \cup \{\varphi \leftarrow: \varphi \in \mathcal{F}\}$ and $\mathcal{P}_2 \cup \{\varphi \leftarrow: \varphi \in \mathcal{F}\}$ have the same stable models.*

Note that, clearly, two strongly equivalent deontic logic programs are also uniformly equivalent.

5.2. Deontic equilibrium logic

Equilibrium logic (Pearce, 2006), and its monotonic base—the logic of here-and-there (HT; Heyting, 1930)—provide a logical foundation for the stable models semantics of logic programs (Gelfond & Lifschitz, 1988), in which several important properties of logic programs under the stable model semantics can be studied. In particular, it can be used to check strong equivalence of programs by checking logical equivalence in HT (cf. Lifschitz et al., 2000).

Our aim in this section is to extend the results in Lifschitz et al. (2000) to deontic logic programs. Therefore, we define a deontic version of HT, dubbed $\mathrm{HT}_{\mathrm{SDL}}$. The key idea is that a deontic logic program can be naturally translated to a set of formulas in $\mathrm{HT}_{\mathrm{SDL}}$, one for each rule of the program, and strong equivalence of two deontic logic programs can be checked by proving logical equivalence of the corresponding sets of formulas. The non-disjunctive case of this construction can be seen as a particular case of the general construction presented in Gonçalves and Alferes (2011).

The language of $\mathrm{HT}_{\mathrm{SDL}}$ is build constructively from the formulas of SDL using the connectives $\bot, \sqcap, \sqcup$ and $\sqsupset$. Negation $\neg$ is introduced as an abbreviation $\neg\delta := (\delta \sqsupset \bot)$. Note that the formulas of SDL act as atoms of the $\mathrm{HT}_{\mathrm{SDL}}$ language.

The semantics of $\mathrm{HT}_{\mathrm{SDL}}$ is a generalisation of the intuitionistic Kripke semantics of HT. A frame for $\mathrm{HT}_{\mathrm{SDL}}$ is a tuple $\langle W, \leq\rangle$ where W is a set of exactly two worlds, say h (here) and t (there) with $h \leq t$. An $\mathrm{HT}_{\mathrm{SDL}}$ interpretation is a frame together with an assignment i that associates to each world a logical theory of SDL, such that $i(h) \subseteq i(t)$. Note the key idea of substituting, in the original definition of HT interpretations, sets of atoms with SDL logical theories. An interpretation is said to be *total* if $i(h) = i(t)$. It is convenient to see an $\mathrm{HT}_{\mathrm{SDL}}$ interpretation as an ordered pair $\langle T^h, T^t\rangle$ such that $T^h = i(h)$ and $T^t = i(t)$ where i is the interpretation's assignment. We define the satisfaction relation between an $\mathrm{HT}_{\mathrm{SDL}}$ interpretation $I = \langle T^h, T^t\rangle$ at a particular world $w \in \{h, t\}$ and an $\mathrm{HT}_{\mathrm{SDL}}$ formula δ recursively:

(1) for $\varphi \in L_{\mathrm{SDL}}$ we have that $\langle T^h, T^t\rangle, w \Vdash \varphi$ if $T^w \vdash_{\mathrm{SDL}} \varphi$;
(2) $I, w \nVdash \bot$;
(3) $I, w \Vdash (\delta_1 \sqcup \delta_2)$ if $I, w \Vdash \delta_1$ or $I, w \Vdash \delta_2$;
(4) $I, w \Vdash (\delta_1 \sqcap \delta_2)$ if $I, w \Vdash \delta_1$ and $I, w \Vdash \delta_2$;
(5) $I, w \Vdash (\delta_1 \sqsupset \delta_2)$ if $\forall_{w \leq w'}$ we have $I, w' \nVdash \delta_1$ or $I, w' \Vdash \delta_2$.

We say that an interpretation $\mathcal{I}$ is a model of an $\mathrm{HT}_{\mathrm{SDL}}$ formula δ if $\mathcal{I}, w \Vdash \delta$ for every $w \in \{h, t\}$. A formula δ is said to be a consequence of a set of formulas Φ, denoted by $\Phi \vdash_{\mathrm{HT}_{\mathrm{SDL}}} \delta$, if for every interpretation $\mathcal{I}$ and every world w we have that $\mathcal{I}, w \Vdash \delta$ whenever $\mathcal{I}, w \Vdash \delta'$ for every $\delta' \in \Phi$. Two sets of formulas Φ_1 and Φ_2 are said to be equivalent if $\Phi_1 \vdash_{\mathrm{HT}_{\mathrm{SDL}}} \psi$ for every $\psi \in \Phi_2$ and $\Phi_2 \vdash_{\mathrm{HT}_{\mathrm{SDL}}} \psi$ for every $\psi \in \Phi_1$.

Definition 17. *An equilibrium model of a set Φ of HT_{SDL} formulas is a total HT_{SDL} interpretation $\langle T, T\rangle$ such that*

(1) $\langle T, T\rangle$ is a model of Φ;
(2) for every SDL theory $T' \subset T$ we have that $\langle T', T\rangle$ is not a model of Φ.

With this notion of a model, equilibrium entailment is defined as follows:

Definition 18. *The equilibrium entailment, $\vDash_E$, over HT_{SDL} formulas is defined for every set $\Phi \cup \{\delta\}$ of HT_{SDL} formulas as follows:*

- *if Φ is non-empty and has equilibrium models then $\Phi \vDash_E \delta$ if every equilibrium model of Φ is an HT_{SDL} model of δ;*
- *if Φ is empty or does not have equilibrium models then $\Phi \vDash_E \delta$ if $\Phi \vdash_{HT_{SDL}} \delta$.*

The above definition of entailment is in line with the usual definition of equilibrium entailment. It assumes the notion of entailment as truth in every equilibrium model, which, as our stable model entailment, is a sceptical notion of entailment. The second condition in the above definition is targeted at those sets of formulas which do not have equilibrium models. Note that a set of formulas without equilibrium models is not necessarily inconsistent, since it can have $\mathrm{HT}_{\mathrm{SDL}}$ models.

Our aim now is to prove that $\mathrm{HT}_{\mathrm{SDL}}$ can be used to prove strong equivalence of deontic logic programs (Theorem 24). For that, we identify a rule r of the form

$$\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n, \mathit{not}\ \delta_1, \ldots, \mathit{not}\ \delta_m$$

of a deontic logic program with the HT$_{\text{SDL}}$ formula $\pi(r)$

$$(\psi_1 \sqcap \ldots \sqcap \psi_n \sqcap \neg\delta_1 \sqcap \ldots \sqcap \neg\delta_m) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k),$$

and a program $\mathcal{P}$ with the set $\pi(\mathcal{P})$ of HT$_{\text{SDL}}$ formulas that correspond to its rules, i.e., $\pi(\mathcal{P}) = \bigcup_{r\in\mathcal{P}} \pi(r)$.

On our way to stating and proving the main result of this section, Theorem 24 below, we present several interesting lemmas.

Lemma 19. *Let $\mathcal{P}$ be a definite deontic logic program. Then, an HT$_{\text{SDL}}$ interpretation $\mathcal{I} = \langle T^h, T^t\rangle$ is a model of $\pi(\mathcal{P})$ iff both T^h and T^t are models of $\mathcal{P}$ and $T^h \subseteq T^t$.*

Proof. Let $\mathcal{P}$ be a definite deontic logic program. An HT$_{\text{SDL}}$ interpretation $\mathcal{I}$ is a model of $\pi(\mathcal{P})$ iff for every rule $\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n \in \mathcal{P}$ and every $w \in \{h, t\}$ we have that $\mathcal{I}, w \Vdash (\psi_1 \sqcap \ldots \sqcap \psi_n) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k)$ iff for every rule $\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n \in \mathcal{P}$ and every $w \in \{h, t\}$ if $\mathcal{I}, w \Vdash \psi_i$ for every $i \in \{1, \ldots, n\}$, then $\mathcal{I}, w \Vdash \varphi_j$ for some $j \in \{1, \ldots, k\}$. This amounts to saying that T^h and T^t both satisfy each rule of $\mathcal{P}$, i.e., T^h and T^t are both models of $\mathcal{P}$. ■

Lemma 20. *Let $\mathcal{P}$ be a deontic logic program. Then, an HT$_{\text{SDL}}$ interpretation $\langle T^h, T^t\rangle$ is a model of $\pi(\mathcal{P})$ iff it is a model of $\pi\left(\frac{\mathcal{P}}{T^t}\right)$.*

Proof. Recall that $\frac{\mathcal{P}}{T^t}$ is obtained from $\mathcal{P}$ by dropping every rule r of $\mathcal{P}$ such that *not* $\psi \in r$ and $\psi \in T^t$ for some SDL formula ψ, and by eliminating all default atoms from the remaining rules.

Suppose first that $\langle T^h, T^t\rangle$ is a model of $\pi(\mathcal{P})$. We aim to prove that it is also a model of $\pi\left(\frac{\mathcal{P}}{T^t}\right)$. A rule $\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n \in \frac{\mathcal{P}}{T^t}$ is obtained from the rule $\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n, \textit{not}\ \delta_1, \ldots, \textit{not}\ \delta_m \in \mathcal{P}$ where $\delta_1, \ldots, \delta_m \notin T^t$. Therefore, we have that $\langle T^h, T^t\rangle \Vdash \neg\delta_i$ for every $i \in \{1, \ldots, m\}$, and we can conclude that $\langle T^h, T^t\rangle \Vdash (\psi_1 \sqcap \ldots \sqcap \psi_n) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k)$ iff $\langle T^h, T^t\rangle \Vdash (\psi_1 \sqcap \ldots \sqcap \psi_n \sqcap \neg\delta_1 \sqcap \ldots \sqcap \neg\delta_m) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k)$. The latter holds because $\langle T^h, T^t\rangle$ is assumed to be a model of $\pi(\mathcal{P})$.

Now assume that $\langle T^h, T^t\rangle$ is a model of $\pi\left(\frac{\mathcal{P}}{T^t}\right)$. We aim to prove that it is also a model of $\pi(\mathcal{P})$. Let $r = \varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n, \textit{not}\ \delta_1, \ldots, \textit{not}\ \delta_m \in \mathcal{P}$. Then we have two cases.

Case 1: There exists some $i \in \{1, \ldots, m\}$ such that $\delta_i \in T^t$. Then, $\langle T^h, T^t\rangle \nVdash \neg\delta_i$ and trivially we have that $\langle T^h, T^t\rangle$ satisfies $\pi(r)$.

Case 2: For every $i \in \{1, \ldots, m\}$ we have that $\delta_i \notin T^t$. Then $\langle T^h, T^t\rangle \Vdash \neg\delta_i$ for every $i \in \{1, \ldots, m\}$. Therefore, $\langle T^h, T^t\rangle \Vdash (\psi_1 \sqcap \ldots \sqcap \psi_n \sqcap \neg\delta_1 \sqcap \ldots \sqcap \neg\delta_m) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k)$ iff $\langle T^h, T^t\rangle \Vdash (\psi_1 \sqcap \ldots \sqcap \psi_n) \sqsupset (\varphi_1 \sqcup \ldots \sqcup \varphi_k)$. The latter holds because $\langle T^h, T^t\rangle$ is assumed to be a model of $\pi\left(\frac{\mathcal{P}}{T^t}\right)$ and $\varphi_1; \ldots; \varphi_k \leftarrow \psi_1, \ldots, \psi_n \in \frac{\mathcal{P}}{T^t}$. So, we can conclude that $\langle T^h, T^t\rangle$ satisfies $\pi(r)$ for every rule r of $\mathcal{P}$ and it is, therefore, a model of $\pi(\mathcal{P})$. ■

Lemma 21. *Let $\mathcal{P}$ be a deontic logic program. Then, an HT$_{\text{SDL}}$ interpretation $\langle T, T\rangle$ is an equilibrium model of $\pi(\mathcal{P})$ iff T is stable model of $\mathcal{P}$.*

Proof. T is a stable model of $\mathcal{P}$ iff T is a stable model of $\frac{\mathcal{P}}{T}$ iff T is a model of $\frac{\mathcal{P}}{T}$ and for every theory $T' \subset T$, T' is a not model of $\frac{\mathcal{P}}{T}$. Using Lemma 19, this can be equivalently restated as $\langle T, T\rangle$ is a model of $\pi\left(\frac{\mathcal{P}}{T}\right)$ and for every theory $T' \subset T$, we have that $\langle T', T\rangle$ is a not model of $\pi\left(\frac{\mathcal{P}}{T}\right)$. By Lemma 20 this is equivalent to the conditions: $\langle T, T\rangle$ is a model of $\pi(\mathcal{P})$ and for every theory $T' \subset T$, $\langle T', T\rangle$ is not a model of $\pi(\mathcal{P})$. By definition, this means that $\langle T, T\rangle$ is an equilibrium model of $\pi(\mathcal{P})$. ■

An immediate consequence of the above lemma is that, for deontic logic programs which have stable models, equilibrium entailment captures stable models entailment.

Corollary 22. *Let $\mathcal{P}$ be a deontic logic program which has stable models. Then, for every SDL formula φ, we have that $\mathcal{P} \vDash_{SM} \varphi$ iff $\pi(\mathcal{P}) \vDash_{E} \varphi$.*

Lemma 23. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two deontic logic programs. Then, the following conditions are equivalent:*

(1) for every deontic logic program $\mathcal{P}$, the sets $\pi(\mathcal{P}_1 \cup \mathcal{P})$ and $\pi(\mathcal{P}_2 \cup \mathcal{P})$ have the same equilibrium models;
(2) $\pi(\mathcal{P}_1)$ and $\pi(\mathcal{P}_2)$ are equivalent in HT_{SDL}.

Proof. First suppose that $\pi(\mathcal{P}_1)$ and $\pi(\mathcal{P}_2)$ are equivalent in HT_{SDL}. Then, it is clear that $\pi(\mathcal{P}_1 \cup \mathcal{P})$ and $\pi(\mathcal{P}_2 \cup \mathcal{P})$ are also equivalent and, in particular, have the same equilibrium models. We now prove the reverse direction by contraposition. Without loss of generality, suppose that $\pi(\mathcal{P}_1)$ has a model $\langle T^h, T^t\rangle$ which is not a model of $\pi(\mathcal{P}_2)$. We show how to find a program $\mathcal{P}$ such that $\langle T^t, T^t\rangle$ is an equilibrium model of one of the sets $\pi(\mathcal{P}_1 \cup \mathcal{P})$ or $\pi(\mathcal{P}_2 \cup \mathcal{P})$ but not an equilibrium model of the other. We have two cases:

Case 1: $\langle T^t, T^t\rangle$ is not a model of $\pi(\mathcal{P}_2)$. It is easy to see that it is a model of $\pi(\mathcal{P}_1)$. We then take $\mathcal{P} = \{\varphi \leftarrow : \varphi \in T^t\}$. Clearly $\langle T^t, T^t\rangle$ is a model of $\pi(\mathcal{P}_1 \cup \mathcal{P})$. Since for every $T \subset T^t$ we have that $\langle T, T^t\rangle$ is not a model of $\pi(\mathcal{P}_1 \cup \mathcal{P})$, we can conclude that $\langle T^t, T^t\rangle$ is an equilibrium model of $\pi(\mathcal{P}_1 \cup \mathcal{P})$. On the other hand $\langle T^t, T^t\rangle$ is not a model of $\pi(\mathcal{P}_2 \cup \mathcal{P})$ since it is not a model of $\pi(\mathcal{P}_2)$.

Case 2: $\langle T^t, T^t\rangle$ is a model of $\pi(\mathcal{P}_2)$. Take $\mathcal{P} = \{\varphi \leftarrow : \varphi \in T^h\} \cup \{\psi \leftarrow \varphi : \varphi, \psi \in T^t \setminus T^h, \varphi \neq \psi\}$. Clearly $\langle T^t, T^t\rangle$ is a model of $\pi(\mathcal{P})$ and it is also a model of $\pi(\mathcal{P}_2 \cup \mathcal{P})$. Let us now prove that it is in fact an equilibrium model. Suppose that there exists $T \subset T^t$ such that $\langle T, T^t\rangle$ is a model of $\pi(\mathcal{P}_2 \cup \mathcal{P})$. By definition of $\mathcal{P}$, we have $T^h \subseteq T$, but $T \neq T^h$ because $\langle T^h, T^t\rangle$ is not a model of $\pi(\mathcal{P}_2)$. Therefore, $T^h \subset T \subset T^t$. If we take $\varphi \in T \setminus T^h$ and $\psi \in T^t \setminus T$ we have that $\langle T, T^t\rangle$ does not satisfy $\varphi \sqsupset \psi$. But this contradicts the fact that $\langle T, T^t\rangle$ is a model of $\pi(\mathcal{P}_2 \cup \mathcal{P})$. It remains to be checked that $\langle T^t, T^t\rangle$ is not an equilibrium model of $\pi(\mathcal{P}_1 \cup \mathcal{P})$. This follows from the straightforward fact that $\langle T^h, T^t\rangle$ is a model of $\pi(\mathcal{P}_1 \cup \mathcal{P})$, along with the fact that $T^h \neq T^t$, because $\langle T^t, T^t\rangle$ is a model of $\pi(\mathcal{P}_2)$ but $\langle T^h, T^t\rangle$ is not. ■

The following is the main theorem of this section. It allows us to prove strong equivalence of two deontic logic programs by checking logical equivalence in HT_{SDL}. Its proof follows immediately from Lemma 23 and Lemma 21.

Theorem 24. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two deontic logic programs. Then the following conditions are equivalent:*

(1) $\mathcal{P}_1$ and $\mathcal{P}_2$ are strongly equivalent;
(2) $\pi(\mathcal{P}_1)$ and $\pi(\mathcal{P}_2)$ are logically equivalent in HT_{SDL}.

6. Embedding input-output logic

In this section we show that the deontic logic programs are expressive enough to embed input-output logic. As a consequence, at the end of the section we show that deontic HT is a sound tool for checking equivalence in the sense of input-output logic. We start by introducing input-output logic.

6.1. Input-output logic

The key idea in input-output logic (IO logic; Makinson & van der Torre, 2000) is to represent norms using pairs of formulas, rather than just with formulas, as 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. In what follows, we denote by L_{CL} the usual set of classical propositional formulas.

Definition 25. *A generating set is a set $G \subseteq L_{\text{CL}} \times L_{\text{CL}}$ of pairs of propositional formulas.*

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. An *input* is a set A of propositional formulas.

Given a generating set G and an input A, 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 consequence of the normative system G given that the formulas of the input A hold. This construction of $G(A)$, however, does not take into account the logical interdependence between formulas. For example, if $G = \{\langle p, q \rangle\}$ and $A = \{p \wedge r\}$ then $q \notin G(A)$. To cope with this interdependency, the so-called *out* operations were defined. The idea is that the *out* operations represent the different ways in which the logical interdependence between formulas can be handled.

The semantics of IO logic is an operational semantics which is parameterised by the choice of these *out* operations. Operation $out(G, A)$ takes a generating set G and an input A and returns an (output) set of formulas. Four natural *out* operations are usually considered: the simple-minded operator out_1, the basic operator out_2, the reusable operator out_3, and the reusable basic operator out_4.

Given a set A of formulas we denote by $Cn(A)$ the set of consequences of A in classical logic. In our terminology we could write $Cn(A) = A^{\vdash_{\text{CL}}}$ where $\vdash_{\text{CL}}$ is the usual consequence relation of classical logic. Recall that a classical theory is a set T such that $Cn(T) = T$. A theory V is said to be *complete* if, for each formula $\varphi \in L_{\text{CL}}$, V contains either φ or its negation $\sim \varphi$. We can now define the *out* operations.

Definition 26. *Given a generating set G and an input A, we define:*

- $out_1(G, A) = Cn(G(Cn(A)))$.
- $out_2(G, A) = \bigcap\{Cn(G(V)) : A \subseteq V \textit{ and } V \textit{ is a complete theory}\}$.
- $out_3(G, A) = \bigcap\{Cn(G(B)) : A \subseteq B,\ B = Cn(B), \textit{and } G(B) \subseteq B\}$.
- $out_4(G, A) = \bigcap\{Cn(G(V)) : A \subseteq V,\ G(V) \subseteq V \textit{ and } V \textit{ is a complete theory}\}$.

For each operator out_n, we can consider the correspondent throughput operator, out_n^+, which 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

$$Id = \{\langle \varphi, \varphi \rangle : \varphi \in L_{\text{CL}}\}.$$

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

Although the above formulation of IO logic already gives a reasonable account of conditional obligations, it is not enough for reasoning with contrary-to-duty situations. Recall that 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 are already being violated. 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. 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.

In order to cope with contrary-to-duty paradoxes, IO logic was extended in Makinson and van der Torre (2001). In this approach, contrary-to-duty situations were connected with problems related to consistency. The issue was how to deal with excessive output, i.e., those cases in which the output was itself inconsistent, or inconsistent with respect to the input. In the last case the input is said to be inconsistent with the output and these cases correspond to contrary-to-duty situations.

Using ideas from non-monotonic reasoning, 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 an input A, $maxfamily(G, A)$ is the set of maximal subsets of G for which the output operator yields a set consistent with the input, i.e., the set

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

The set $outfamily(G, A)$ collects the outputs of the elements 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.

The framework of IO logic allows, in a straightforward way, the definition of the important notion of equivalence between generating sets.

Definition 27. *Two generating sets G and G' are equivalent if, for every set of inputs A, we have that $outfamily(G, A) = outfamily(G', A)$.*

6.2. Embedding

We now present an embedding of IO logic in deontic logic programs. The results presented here, whose proofs can be found in the Appendix, can be seen as a strengthening of the existing weak connection drawn in Makinson and van der Torre (2000) between IO logic and Reiter's default logic.

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 28. *Let G be a generating set, A an input consistent with the output, and φ and ψ classical propositional formulas.*

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

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

(2) Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\psi\ ; \sim \psi \leftarrow :\ \psi, \sim \psi \in L_{CL} \backslash_A\}$.

Then, $out_2(G, A) = \{\varphi : \mathcal{P}_2 \vDash_{SM} \mathbf{O}(\varphi)\}$.

(3) 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, $out_3(G, A) = \{\varphi : \mathcal{P}_3 \vDash_{SM} \mathbf{O}(\varphi)\}$.

(4) Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\psi\ ; \sim \psi \leftarrow :\ \psi, \sim \psi \in L_{CL} \backslash_A\}$.
Then, $out_4(G, A) = \{\varphi : \mathcal{P}_4 \vDash_{SM} \mathbf{O}(\varphi)\}$.

(5) For each $i \in \{1, \dots, 4\}$, *let* $\mathcal{P}_i^+ = \mathcal{P}_i \cup \{\mathbf{O}(\varphi) \leftarrow \varphi\ :\ \varphi \in L_{CL}\}$.
Then, $out_i^+(G, A) = \{\varphi : \mathcal{P}_i^+ \vDash_{SM} \mathbf{O}(\varphi)\}$.

Note that for the embedding in the lemma, standard normal logic programs are not enough. Not only they do not consider obligations but, equally importantly, they do not allow for complex formulas in the heads and bodies of rules. Bear in mind that φ and ψ can be any classical propositional formula. However, in this lemma we only need 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 flavour and, as we will see below, default negation plays a fundamental role in the embedding.

In Makinson and van der Torre (2000) it was shown that, given a generating set G, an input A, 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 we can capture the entire $outfamily$ using our stable models semantics.

Theorem 29. *Let G be a generating set and A an input, 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* $\psi \in L_{CL}$:

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

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 as in 1):

$$\mathcal{P}_2 = \mathcal{P}_1 \cup \{\psi \,;\, \sim \psi \leftarrow :\ \psi, \sim \psi \in L_{CL} \backslash A\}.$$

Then, taking out_2^+ *as the out operator, we have*

$$outfamily(G, A) = \bigcup_{T \in SM(\mathcal{P}_2)} \{\varphi : \mathbf{O}(\varphi) \in T\}.$$

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

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

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

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

(4) Consider the deontic logic program, over an extended language as in 3):

$$\mathcal{P}_4 = \mathcal{P}_3 \cup \{\psi \,;\, \sim \psi \leftarrow :\ \psi, \sim \psi \in L_{CL} \backslash A\}.$$

Then, taking out_4^+ *as the out operator, we have*

$$outfamily(G, A) = SM(\mathcal{P}_4)_{|L_{CL}}.$$

One could wonder why, in the above theorem, and in the cases of out_3^+ and out_4^+, we did not need to consider deontic logic programs with deontic operators. The reason is that if we admit throughput (i.e., inputs being 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. As noted in Makinson and van der Torre (2007), this 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 these cases obligations are not used, standard logic programming is not enough for the above embeddings. The reason is that, as already noted above after Lemma 28, we need to consider complex propositional formulas in both the body and the 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. Next, it will be interesting to consider what additional features deontic logic programming can 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 for embedding IO logic are not very expressive when compared with the expressivity of a normal deontic logic program. Additionally, in deontic logic programming we have an explicit use of default negation, which is fundamental to model exceptions. Moreover, deontic logic programs can have complex deontic formulas not only in the head, but also in the body of a rule. This is fundamental for modelling violations of obligations and specifying 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. The following corollary, which is an immediate consequence of the embedding theorem, draws a tight connection between equivalence of generating sets in IO logic and the notion of uniform equivalence of deontic logic programs.

Corollary 30. *Suppose that we take out_3^+ as the out operator. Then, two generating sets G and G' are equivalent if their corresponding deontic logic programs,*

$$\begin{aligned}\mathcal{P}_G &= \{\psi \leftarrow \varphi, not\ \overline{\psi} : \langle \varphi, \psi \rangle \in G\} \cup \{\overline{\psi} \leftarrow not\ \psi : \psi \in L_{CL}\}, \text{ and}\\ \mathcal{P}_{G'} &= \{\psi \leftarrow \varphi, not\ \overline{\psi} : \langle \varphi, \psi \rangle \in G'\} \cup \{\overline{\psi} \leftarrow not\ \psi : \psi \in L_{CL}\},\end{aligned}$$

are uniformly equivalent.

As we saw in Section 5, we can check strong equivalence of deontic logic programs using logical equivalence in deontic HT. Interestingly, this immediately gives a sound tool to check equivalence of generating sets.

Corollary 31. *Suppose that we take out_3^+ as the out operator. Then, two generating sets G and G' are equivalent if the sets of HT_{SDL} formulas corresponding to the programs*

$$\begin{aligned}\mathcal{P}_G &= \{\psi \leftarrow \varphi, not\ \overline{\psi} : \langle \varphi, \psi \rangle \in G\} \cup \{\overline{\psi} \leftarrow not\ \psi : \psi \in L_{CL}\}, \text{ and}\\ \mathcal{P}_{G'} &= \{\psi \leftarrow \varphi, not\ \overline{\psi} : \langle \varphi, \psi \rangle \in G'\} \cup \{\overline{\psi} \leftarrow not\ \psi : \psi \in L_{CL}\},\end{aligned}$$

are logically equivalent in HT_{SDL}.

Note that the above test is sound but not complete. To be more precise, if the sets of $\mathrm{HT}_{\mathrm{SDL}}$ formulas corresponding to the programs $\mathcal{P}_G$ and $\mathcal{P}_{G'}$ are not logically equivalent in $\mathrm{HT}_{\mathrm{SDL}}$, then we cannot conclude anything about the equivalence of G and G'. Note also that, by modifying the program construction according to Theorem 29, the above two corollaries hold for the other out operations. We illustrated the corollaries using out_3^+ only because it is the one with the simplest program construction.

We end this section by contrasting the use of deontic logic programs with IO logic in the contrary-to-duty situation of Example 14.

Example 32. *Recall the normative statement given in Example 14.*

The following is the representation proposed in Makinson and van der Torre (2007) of that statement using IO logic:

$$\langle \boldsymbol{t},\ \sim (dog \vee fence)\rangle \qquad \langle dog,\ \ fence \wedge warningSign\rangle.$$

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 because it includes $\sim dog$, but is itself also inconsistent because it includes both $fence$ and $\sim fence$.*

The use of constrained output solves this particular problem. In fact, we have that:

$$\begin{aligned}&maxfamily(G, A) = \{\{\langle dog,\ fence \wedge warningSign\rangle\}\}\\ &outfamily(G, A) = \{Cn(fence \wedge warningSign)\}.\end{aligned}$$

Intuitively, since dog is the case, the conditional norm $\langle \boldsymbol{t},\ \sim (dog \vee fence)\rangle$ 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 pointed out in Makinson and van der Torre (2007), this strategy is very sensible in terms of how the generating set is written, but in some cases it discards too much of the output. As an example, suppose that we only *consider the first conditional norm $\langle \boldsymbol{t}, \sim (dog \vee fence)\rangle$. If dog is the case then, surprisingly, $outfamily(G, A)$ only contains the set of tautologies and therefore does not include $\sim fence$.*

The motivation behind the idea of constraining the output to deal with contrary-to-duty situations is, as argued in Makinson and van der Torre (2001), 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 to (and can!) reason about the violation of obligations. Consider that in the cottage example we add rules for applying sanctions in case of violations, i.e., we augment the above normative system with the rules

$$\mathbf{O}(fineD) \leftarrow \mathbf{O}(\sim dog), dog$$
$$\mathbf{O}(fineF) \leftarrow \mathbf{O}(\sim fence), fence.$$

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 if we were to assume that $\mathbf{O}(\sim dog)$ should not follow from the normative system when dog is the case.

7. Conclusions and future work

In this paper we have introduced a framework for representing and reasoning about normative systems which combines the expressivity of standard deontic logic with non-monotonic logic programs. We have defined a stable model semantics and studied the problem of equivalence between normative systems. We have also proved that an important part of IO logic can be embedded in our framework.

Although we have defined a stable model semantics, the approach used in this paper enables a well-founded semantics to be easily defined by making use of logical foundations of the well-founded semantics (Cabalar, Odintsov, & Pearce, 2006). As usual, this could be seen as a sound sceptical approximation of the stable model semantics.

Being declarative, our normative framework could easily be integrated into normative multi-agent systems that use declarative languages for modelling norms (such as, for example, Hubner, Sichman, & Boissier, 2007; Tinnemeier, Dastani, & Meyer, 2009). This would allow an increase of expressivity in their norm languages. Although this is not usually the main focus of such systems, the need for more expressive declarative languages for representing norms was realised (for example, in Tinnemeier, Dastani, Meyer, & van der Torre, 2009).

Our approach could easily be adapted to combine non-monotonic logic programs with extensions of SDL. In fact, a careful look at the results and proofs in this paper reveals that the crucial point is to guarantee that the set of theories of the deontic logic, which is then used as an interpretation of the (deontic) logic program, forms a complete lattice. This is what assures the existence, for every set of formulas A, of the smallest theory, $A^{\vdash_{\mathrm{SDL}}}$, which contains A. But, it can be shown that this is guaranteed for any deontic logic DL for which $\vdash_{DL}$ is a so-called Tarskian consequence relation (Wójcicki, 1988), i.e., it satisfies, for every $T \cup \Phi \cup \{\varphi\} \subseteq L_{DL}$, *Reflexivity*: if $\varphi \in T$ then $T \vdash_{\mathcal{L}} \varphi$; *Cut*: if $T \vdash_{DL} \varphi$ for all $\varphi \in \Phi$, and $\Phi \vdash_{DL} \psi$ then $T \vdash_{DL} \psi$; *Weakening*: if $T \vdash_{DL} \varphi$ and $T \subseteq \Phi$ then $\Phi \vdash_{DL} \varphi$. Interesting examples of extensions of SDL include dyadic deontic logics and temporal extensions of SDL, the latter of which would allow some notion of time to be incorporated into the framework.

Interesting topics for future work include the concepts of redundancy and consistency, the former of which could be defined using the notion of strong equivalence.

Definition 33. *Let $\mathcal{P}$ be a deontic logic program and let r be a rule of $\mathcal{P}$. Then, r is redundant with respect to program $\mathcal{P}$ if $\mathcal{P} \equiv_s \mathcal{P} \setminus \{r\}$.*

Redundancy is important for normative systems simplification. If a rule is redundant with respect to a normative system then we can safely remove it without affecting the semantics of the normative system.

Consistency is also an important property of a normative system. When can we say that a normative system is consistent with a set of facts? A solution could be found along the lines of what is proposed in Makinson and van der Torre (2000) for IO logic.

Definition 34. *Given a normative system $\mathcal{N}$ and a consistent set of facts $\mathcal{F}$, we say that $\mathcal{N}$ is consistent with $\mathcal{F}$ if the deontic logic program $\mathcal{N} \cup \mathcal{F}$ has a consistent stable model. A normative system $\mathcal{N}$ is* consistent *if for any consistent set of facts $\mathcal{F}$, $\mathcal{N}$ is consistent with $\mathcal{F}$.*

Let us clarify the definition with the help of an example. Consider the first normative system presented in Example 14, i.e., where $\mathcal{N}$ is:

$$\begin{aligned} \mathbf{O}(\sim dog \wedge \sim fence) &\leftarrow \\ \mathbf{O}(fence \wedge warningSign) &\leftarrow dog. \end{aligned}$$

Clearly, there is a conflict between the two heads of the rules whenever dog is the case. In fact, if we take the consistent set $\mathcal{F} = \{dog\}$, the program $\mathcal{N} \cup \mathcal{F}$ has only one stable model—the set of all SDL formulas, which is not a consistent set. Therefore, according to our definition, this normative system is not consistent with $\mathcal{F} = \{dog\}$.

Consider now the second normative system of Example 14, i.e., where $\mathcal{N}$ is:

$$\begin{aligned} \mathbf{O}(\sim dog) &\leftarrow \\ \mathbf{O}(\sim fence) &\leftarrow not\ dog \\ \mathbf{O}(fence \wedge warningSign) &\leftarrow dog. \end{aligned}$$

In this case, although there is a conflict between the head of the rules, it is clear that the bodies of these rules cannot be true at the same time. Therefore, $\mathcal{N}$ is a normative system which is consistent not only with the set $\mathcal{F} = \{dog\}$, but also with any consistent set of facts containing only formulas without deontic operators. It would be nice to further explore these notions of redundancy and consistency.

With the aim of developing an implementation of our framework, we are currently working on algorithms that make a modular use of answer-set solvers and SDL reasoners, where the SDL reasoners are used just as oracles within the answer-set solver. Regarding complexity, it is clear that the combination of SDL with non-monotonic logic programs increases the complexity of SDL alone, as stable models add, as usual, one extra level of non-determinism. It is also clear that it increases the complexity of using answer-sets alone (recall that the validity problem in SDL is PSPACE-complete). This extra complexity is not surprising given the expressivity of the language of deontic logic programs. Recall that a deontic logic program can have any SDL formula in the head and body of its rules. In some particular applications, however, there is no need for this general expressivity, and, in those cases we can play the usual game between expressivity and complexity. We are currently studying specific classes that do not increase the complexity of stable models alone, with promising preliminary results (Gonçalves & Alferes, in press). For example, by extending normal logic programs with deontic operators applied to literals (atoms or classical negation of atoms), and classical negation of deontic operators, the complexity of stable models of normal programs is kept. Moreover, we are able to compute the stable models of such restricted deontic logic programs using just an answer-set solver, i.e., without using an SDL reasoner.

However, the study of which restricted classes of deontic logic programs have enough expressivity for specifying complex normative systems is left for future work.

Acknowledgements

The second author was supported by FCT under the postdoctoral grant SFRH/BPD/47245/2008. The work was partially supported by projects ERRO – PTDC/EIA-CCO/121823/2010, and ASPEN – PTDC/EIA-CCO/110921/2009.

Notes

1. In this work we use the word *fact* with its usual meaning in the literature of normative multi-agent systems (Tinnemeier, Dastani, & Meyer, 2009), i.e., as a formula partially describing the state of the environment or the state of the normative system. This should not be confused with the logic programming notion of *fact*.
2. Note that our aim is not to enter into a philosophical discussion about the notion of permission. We just want to point out that we can represent these two versions of permission. We refer to von Wright (1963) for a philosophical discussion.

References

Alberti, M., Gomes, A. S., Gonçalves, R., Leite, J., & Slota, M. (2011). Normative systems represented as hybrid knowledge bases. In J. Leite, P. Torroni, T. Ågotnes, G. Boella, & L. W. N. van der Torre (Eds.), *Computational Logic in Multi-Agent Systems – 12th International Workshop, CLIMA XII, Barcelona, Spain, July 17–18, 2011. Proceedings* (pp. 330–346). New York, NY: Springer.

Alberti, M., Knorr, M., Gomes, A. S., Leite, J., Gonçalves, R., & Slota, M. (2012). Normative systems require hybrid knowledge bases. In W. van der Hoek, L. Padgham, V. Conitzer & M. Winikoff (Eds.), *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4–8, 2012 (3 Volumes)* (pp. 1425–1426). New York, NY: ACM Press.

Boella, G., Governatori, G., Rotolo, A., & van der Torre, L. W. N. (2010). A logical understanding of legal interpretation. In F. Lin, U. Sattler, & M. Truszczynski (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9–13, 2010*. Menlo Park, CA: AAAI Press. (Retrieved, 20 April 2013, from `http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1379`)

Brewka, G. (1996). Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, *4*, 19–36.

Cabalar, P., Odintsov, S. P., & Pearce, D. (2006). Logical foundations of well-founded semantics. In P. Doherty, J. Mylopoulos & C. A. Welty (Eds.), *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2–5, 2006* (pp. 25–35). Menlo Park, CA: AAAI Press.

Chellas, B. (1980). *Modal logic: An introduction*. Cambridge: Cambridge University Press.

Chisholm, R. (1963). Contrary-to-duty imperatives and deontic logic. *Analysis*, *24*, 33–36.

Eiter, T., & Fink, M. (2003). Uniform equivalence of logic programs under the stable model semantics. In C. Palamidessi (Ed.), *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9–13, 2003, proceedings* (pp. 224–238). Berlin: Springer.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In R. A. Kowalski & K. A. Bowen (Eds.), *Logic Programming, proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15–19, 1988* (pp. 1070–1080). Cambridge, MA: MIT Press.

Gonçalves, R., & Alferes, J. J. (2010). Parametrized logic programming. In T. Janhunen & I. Niemelä (Eds.), *Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13–15, 2010. Proceedings* (pp. 182–194). New York, NY: Springer.

Gonçalves, R., & Alferes, J. J. (2011). Parametrized equilibrium logic. In J. P. Delgrande & W. Faber (Eds.), *Logic Programming and Nonmonotonic Reasoning – 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16–19, 2011. Proceedings* (pp. 236–241). Berlin: Springer.

Gonçalves, R., & Alferes, J. J. (in press). Deontic logic programs. In M. Gini, O. Shehory, T. Ito, & C. Jonker (Eds.), *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2013, Saint Paul, Minnesota, USA, May 6–10, 2013*. New York, NY: ACM Press.

Governatori, G., & Rotolo, A. (2008). BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, *17*, 36–69.

Heyting, A. (1930). *Die formalen regeln der intuitionistischen logik*. Berlin: Akademie. (Reprinted in Kreiser, K., & Berka, L. (1986). *Logik-texte: Kommentierte auswahl zur geschichte der modernen logik*. Berlin: Akademie.)

Hilpinen, R. (1981). *Deontic logic: Introductory and systematic readings*. Dordrecht: Kluwer Academic.

Horty, J. F. (1993). Deontic logic as founded on nonmonotonic logic. *Annals of Mathematics and Artificial Intelligence*, *9*, 69–91.

Hubner, J., Sichman, J., & Boissier, O. (2007). Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, *1*, 370–395.

Lewis, D. (1999). *Semantic analyses for dyadic deontic logic*. Cambridge: Cambridge University Press.

Lifschitz, V., Pearce, D., & Valverde, A. (2000). Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, *2*, 526–541.

Makinson, D., & van der Torre, L. W. N. (2000). Input-output logics. *Journal of Philosophical Logic*, *29*, 383–408.

Makinson, D., & van der Torre, L. W. N. (2001). Constraints for input/output logics. *Journal of Philosophical Logic*, *30*, 155–185.

Makinson, D., & van der Torre, L. W. N. (2007). What is input/output logic? Input/output logic, constraints, permissions. In G. Boella, L. W. N. van der Torre, & H. Verhagen (Eds.), *Normative Multi-agent Systems, 18.03.–23.03.2007*. Dagstuhl: IBFI.

McCarty, L. T. (1994). Defeasible deontic reasoning. *Fundamenta Informaticae*, *21*, 125–148.

Nute, D. (1997). *Defeasible deontic logic*. New York, NY: Springer.

Pearce, D. (2006). Equilibrium logic. *Annals of Mathematics and Artificial Intelligence*, *47*, 3–41.

Pearce, D., & Valverde, A. (2004). Uniform equivalence for equilibrium logic and logic programs. In V. Lifschitz & I. Niemelä (Eds.), *Logic Programming and Nonmono- tonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6–8, 2004, proceedings* (pp. 194–206). Berlin: Springer.

Prakken, H., & Sartor, G. (1997). Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, *7*, 25–75.

Prakken, H., & Sergot, M. J. (1996). Contrary-to-duty obligations. *Studia Logica*, *57*, 91–115.

Ryu, Y. U., & Lee, R. M. (1993). Defeasible deontic reasoning: A logic programming model. In J.-J. C. Meyer & R. J. Wieringa (Eds.), *Deontic logic in computer science: Normative system specification* (pp. 225–241). Chichester: John Wiley & Sons.

Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., & Cory, H. T. (1986). The British Nationality Act as a logic program. *Communications of the ACM*, *29*, 370–386.

Tinnemeier, N. A. M., Dastani, M., & Meyer, J.-J. C. (2009). Roles and norms for programming agent organizations. In C. Sierra, C. Castelfranchi, K. S. Decker & J. S. Sichman (Eds.), *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10–15, 2009, Vol. 1* (pp. 121–128). New York, NY: ACM Press.

Tinnemeier, N. A. M., Dastani, M., Meyer, J.-J. C., & van der Torre, L. W. N. (2009). Programming normative artifacts with declarative obligations and prohibitions. In: R. B. Yates, J. Lang, S. Mitra & S. Parsons (Eds.), *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009, Milan, Italy, 15–18 September 2009* (pp. 145–152). Washington, DC: IEEE Computer Society.

van der Torre, L. W. N. (2003). Contextual deontic logic: Normative agents, violations and independence. *Annals of Mathematics and Artificial Intelligence*, *37*, 33–63.

von Wright, G. H. (1951). Deontic logic. *Mind*, *60*, 1–15.
von Wright, G. H. (1963). *Norm and action: A logical enquiry*. London: Routledge & Kegan Paul.
Wójcicki, R. (1988). *Theory of logical calculi*. Dordrecht: Kluwer Academic.

Appendix

Proof of theorem 28. Let us prove the result for out_1 and out_3. The other cases can be proved similarly. For each case below we prove the two inclusions. First, recall that, since $\mathcal{P}_1$ and $\mathcal{P}_3$ are definite non-disjunctive deontic logic programs, they both have a least model, i.e., $\mathcal{P}_1$ and $\mathcal{P}_3$ both have a set of minimal models with only one element. We denote them by $S_{\mathcal{P}_1}$ and $S_{\mathcal{P}_3}$, respectively. Therefore, for every $\varphi \in L_{\text{SDL}}$ we have that $\mathcal{P}_1 \vDash_{\text{SM}} \delta$ is equivalent to $\delta \in S_{\mathcal{P}_1}$, and that $\mathcal{P}_3 \vDash_{\text{SM}} \delta$ is equivalent to $\delta \in S_{\mathcal{P}_3}$.

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 an 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 CL-theory, given the fact that $S_{\mathcal{P}_1}$ is an 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 $\Phi = (\{\mathbf{O}(\psi) : \psi \in out_1(G, A)\} \cup A)^{\vdash_{\text{SDL}}}$ be an 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 $\{\psi : \mathbf{O}(\psi) \in S_{\mathcal{P}_1}\} \subseteq \{\psi : \mathbf{O}(\psi) \in \Phi\}$. Thus, 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_{CL} of all classical formulas of Φ, i.e., those formulas that do not involve the obligation operator. It is easy to prove that Φ_{CL} is a CL-theory and, clearly, $A \subseteq \Phi_{CL}$. 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 $G(\Phi_{CL}) \subseteq \Phi_{CL}$. Then, we have that $G(\Phi_{CL}) \subseteq \{\varphi : \mathbf{O}(\varphi) \in \Phi\}$ 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 $Cn(G(\Phi_{CL})) \subseteq Cn(\{\varphi : \mathbf{O}(\varphi) \in \Phi\})$. But, since Φ is an SDL theory, it is easy to prove that $\{\varphi : \mathbf{O}(\varphi) \in \Phi\}$ is a CL-theory, and, therefore, $Cn(\{\varphi : \mathbf{O}(\varphi) \in \Phi\}) = \{\varphi : \mathbf{O}(\varphi) \in \Phi\}$.

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

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 CL-theories T such that $A \subseteq T$ and $G(T) \subseteq T$. Consider the SDL theory $\Phi_T = (\{\mathbf{O}(\psi) : \psi \in Cn(G(T))\} \cup T)^{\vdash_{\text{SDL}}}$ obtained from $T \in \mathcal{M}$. Using SDL reasoning, it is not hard to see that $(\Phi_T)_{CL}$ is a CL-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 $\{\varphi : \mathbf{O}(\varphi) \in S_{\mathcal{P}_3}\} \subseteq \bigcap_{T \in \mathcal{M}} \{\varphi : \mathbf{O}(\varphi) \in \Phi_T\} = \bigcap_{T \in \mathcal{M}} Cn(G(T)) = out_3(G, A)$. ■

Proof of Theorem 29. Here we only prove condition 3), i.e., the condition for out_3^+. The proofs of the other conditions are similar or even simpler than 3) and can be easily adapted.

We aim to prove that $outfamily(G, A) = SM(\mathcal{P}_3)_{|L_{\text{CL}}}$. We do this by proving the two inclusions separately. First, we prove that $outfamily(G, A) \subseteq SM(\mathcal{P}_3)_{|L_{\text{CL}}}$. 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 $out_3^+(H, A) = \Phi_{|L_{\text{CL}}}$. 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 $\Phi_{|L_{\text{CL}}} \subseteq out_3^+(H, A)$. To prove the converse inclusion just note that $\Phi_{|L_{\text{CL}}}$ 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 $out_3^+(H, A) \subseteq \Phi_{|L_{\text{CL}}}$.

Let us now prove that $SM(\mathcal{P}_3)_{|L_{\text{CL}}} \subseteq outfamily(G, A)$. 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) $T_{|CL} = out_3^+(H, A)$; and (2) H is maximal such that $out_3^+(H, A)$ is consistent. First, 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 $T_{|CL} = out_3^+(H, A)$. 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 $T_{|CL} = out_3^+(H, A)$ 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 $T_{|CL} \subseteq out_3^+(H, A)$. 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 is closed under H. Since $out_3^+(H, A)$ is the minimal one, we can conclude that $out_3^+(G, A) \subseteq T_{|CL}$. Therefore, we can conclude that $T_{|CL} = out_3^+(H, A)$.

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 $T_{|CL} = out_3^+(H, A) \subseteq out_3^+(H', A)$. 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. ■