# Deontic Logic Programs

# (Extended Abstract)

Ricardo Gonçalves
rjrg@fct.unl.pt

José Júlio Alferes
jja@fct.unl.pt

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

## ABSTRACT

Deontic logic programming (DLP) is a framework combining deontic logic and non-monotonic logic programming, and it is useful to represent and reason about normative systems. In this paper we propose an implementation for reasoning in DLP that combines, in a modular way, a reasoner for deontic logic with a reasoner for stable model semantics.

## Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

## General Terms

Languages, Theory, Algorithms

## Keywords

Norms, Knowledge representation, Environments, Social and organisational structure, Logic-based approaches and methods, Design languages for agent systems

## Extended Abstract

Deontic logic programming (DLP) [5, 6] is a framework combining Standard Deontic Logic (SDL) [9, 2] and Logic Programming (LP) [7], for representing and reasoning about normative systems. It has a rich language, allowing complex SDL formulas to appear in the body and head of LP rules, combined with the use of default negation. Moreover, DLPs have a purely declarative semantics, stemming from the stable model semantics (SMS) of logic programs [4]. The language obtained is quite expressive and can be shown to embed extant approaches such as input-output logic [8]. Our aim in this paper is to propose an implementation of DLPs that combines, in a modular way, an SDL reasoner with an ASP reasoner for SMS. A DLP is composed by rules that resemble usual logic program rules, but where complex SDL formulas appear in the place where only atoms were allowed

*Definition 1.* A *deontic logic program* is a set of rules
$\varphi \leftarrow \psi_1, ..., \psi_n, not\ \delta_1, ..., not\ \delta_m$
where each of $\varphi, \psi_1, ..., \psi_n, \delta_1, \ldots, \delta_m$ is an $SDL$ formula.

**Appears in:** *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.


As usual, the symbol $\leftarrow$ represents rule implication, "," represents conjunction and *not* represents default negation. The above rule has the usual reading that $\varphi$ should hold whenever $\psi_1, ..., \psi_n$ hold and $\delta_1, ..., \delta_m$ are not known to hold. Given a DLP $\mathcal{P}$ we denote by $frm(\mathcal{P})$ (resp. $hd(\mathcal{P})$) the set of SDL formulas appearing in $\mathcal{P}$ (resp. in the head of a rule in $\mathcal{P}$). The semantics of DLPs (see [5]) is a SMS obtained by using SDL logical theories (sets of SDL formulas closed under SDL consequence) as interpretations, thus accounting for the interdependency between the SDL formulas appearing in the rules. In it, a stable model $T$ is a fixpoint of a $\Gamma$ operator, defined as the least theory (which always exists) of a definite program obtained by deleting all rules with *not* $\phi$ such that $\phi \in T$, and deleting all remaining *not*s. An SDL formula $\varphi$ is a consequence of $\mathcal{P}$, written $\mathcal{P} \models_{\text{SM}} \varphi$, if $\varphi$ belongs to every stable model of $\mathcal{P}$.

Our first result states that, although the number of SDL logical theories is infinite, the number of stable models of a finite DLP is finite. Using that, we can prove decidability.

THEOREM 1. *Let $\mathcal{P}$ be a finite DLP and $\varphi$ any SDL formula. Then, $\mathcal{P}$ has finitely many stable models, and it is decidable the problem of checking if $\mathcal{P} \models_{SM} \varphi$ is the case.*

Our aim is precisely to propose an implementation of a reasoner for DLPs which combines in a modular way a reasoner for SMS (such as Clasp [3]) and for SDL (such as the KED SDL solver [1]). This modularity is fundamental since it allows using the large body of successful research done in the area of SMS implementation and answer set programming.

Let $\mathcal{P}$ be a DLP. Consider the following normal logic program $\mathcal{P}^{\text{N}}$ obtained from $\mathcal{P}$:

$$\mathcal{P}^{\text{N}} = \mathcal{P} \cup \{\varphi \leftarrow \psi_1, \ldots, \psi_n : \{\psi_1, \ldots, \psi_n\} \subseteq hd(\mathcal{P}) \text{ and } \varphi \in frm(\mathcal{P}) \setminus_{\{\psi_1,\ldots,\psi_n\}} \text{ and } \{\psi_1, \ldots, \psi_n\} \vdash_{\text{SDL}} \varphi\}.$$

$\mathcal{P}^{\text{N}}$ is a normal logic program because the SDL formulas in it are to be considered as normal logic programs atoms.

The key idea underlying the construction of $\mathcal{P}^{\text{N}}$, in order to enforce the interdependency between deontic formulas (which in $\mathcal{P}^{\text{N}}$ are just atoms), is to enrich $\mathcal{P}^{\text{N}}$ with rules that represent the possible deontic reasoning occurring with the formulas of $\mathcal{P}$. We denote by $AS(\mathcal{P}^{\text{N}})$ the set of stable models of $\mathcal{P}^{\text{N}}$, viewed as a normal logic program. The following presents a finite representation of each of the stable models of $\mathcal{P}$, where $A^{\vdash_{\text{SDL}}}$ is the closure of $A$ under $\vdash_{\text{SDL}}$.

THEOREM 2. *Given a deontic logic program $\mathcal{P}$, we have that*

$$SM(\mathcal{P}) = \{A^{\vdash_{SDL}} : A \in AS(\mathcal{P}^{N})\}.$$

Our aim now is to compute these finite representations of the stable models of $\mathcal{P}$. We sketch an algorithm that, given a finite DLP $\mathcal{P}$, returns a normal logic program $\mathcal{P}^{\text{alg}}$.

```
input:  finite deontic logic program P
set i = 1; k = lenght(hd(P));
P^alg := P ∪ {φ ← : φ ∈ frm(P) and ⊢_SDL φ}
while i ≤ k
  for each subset A = {δ_1, ..., δ_i} of hd(P) of size i
    if A ⊢_SDL ⊥ (* A is inconsistent *)
      then for each φ ∈ frm(P) \ A
      add φ ← δ_1, ..., δ_i to P^alg unless
      there is φ ← ψ_1, ..., ψ_n ∈ P^alg with {ψ_1, ..., ψ_n} ⊆ A
    else for each φ ∈ frm(P) \ A
      if propSymb(A) ∩ propSymb(φ) = ∅
       then do nothing
      else if φ ← ψ_1, ..., ψ_n ∈ P^alg with {ψ_1, ..., ψ_n} ⊆ A
       then do nothing
      else if A ⊢_SDL φ
       then add φ ← δ_1, ..., δ_i to P^alg
  i=i+1
return P^alg
```

The above algorithm is an improvement of the definition of $\mathcal{P}^{\text{N}}$, in the sense that it prunes some search paths using well-known properties of both the area of logic programming and of the area of deontic logic. Given these improvements, the algorithm for constructing $\mathcal{P}^{\text{alg}}$ does not, in general, return exactly $\mathcal{P}^{\text{N}}$. One can readily see that $\mathcal{P}^{\text{alg}} \subseteq \mathcal{P}^{\text{N}}$. As expected, the extra rules of $\mathcal{P}^{\text{N}}$ are redundant.

PROPOSITION 1. *Given a finite deontic logic program $\mathcal{P}$, we have that $AS(\mathcal{P}^{alg}) = AS(\mathcal{P}^{N})$.*

The above proposition allows the use of the $\mathcal{P}^{\text{alg}}$ algorithm to construct a finite representation of the stable models of $\mathcal{P}$. This can be done by constructing the normal logic program $\mathcal{P}^{\text{alg}}$ from $\mathcal{P}$ and then calculating its stable models. The former can be done using a SDL reasoner and the later can be done using a SMS reasoner. Note that, as we aimed, this construction is modular in the use of the two reasoners.

Regarding complexity, it should be clear that the use of DLPs, with default negation, increases the complexity of the SDL alone. This comes from the fact that the SMS, with default negation, adds, as usual, one extra level of non-determinism. From the point of view of LP there is also an exponential increasing in the complexity. First, to construct $\mathcal{P}^{\text{N}}$ and $\mathcal{P}^{\text{alg}}$ we need to query a SDL oracle an exponential number of times. Moreover, we need to compute the stable models of $\mathcal{P}^{\text{alg}}$ which, in the extreme case, can have exponentially more rules than $\mathcal{P}$. This extra complexity is not surprising given the expressivity of the language of DLPs. Recall that a DLP 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 now study some classes of restricted DLPs, which may well have the necessary expressivity for modeling non-trivial scenarios.

PROPOSITION 2. *Let $\mathcal{P}$ be a DLP with only atoms and deontic operators applied to atoms. Then $\mathcal{P}^{alg} = \mathcal{P}$.*

Another interesting example is the case of DLPs that only contain literals, obligation applied to literals and negation of obligation applied to literals. Contrarily to the previous case, this language is expressive enough to capture the notion of permission: recall that $\mathbf{P}(p) \equiv_{SDL} \neg\mathbf{O}(\neg p)$.

PROPOSITION 3. *Let $\mathcal{P}$ be a finite DLP with only literals, obligation applied to literals and negations of obligations applied to literals. Then*

$$\mathcal{P}^{alg} = \mathcal{P} \cup \{\varphi \leftarrow \bot : \varphi \in frm(\mathcal{P})\} \cup$$
$$\{\bot \leftarrow p, \neg p : \{p, \neg p\} \subseteq hd(\mathcal{P})\} \cup$$
$$\{\bot \leftarrow \mathbf{O}(p), \mathbf{O}(\neg p) : \{\mathbf{O}(p), \mathbf{O}(\neg p)\} \subseteq hd(\mathcal{P})\} \cup$$
$$\{\bot \leftarrow \mathbf{O}(\ell), \neg\mathbf{O}(\ell) : \{\mathbf{O}(\ell), \neg\mathbf{O}(\ell)\} \subseteq hd(\mathcal{P})\} \cup$$
$$\{\neg\mathbf{O}(\neg p) \leftarrow \mathbf{O}(p) : \mathbf{O}(p) \in hd(\mathcal{P}) \text{ and } \neg\mathbf{O}(\neg p) \in frm(\mathcal{P})\} \cup$$
$$\{\neg\mathbf{O}(p) \leftarrow \mathbf{O}(\neg p) : \mathbf{O}(\neg p) \in hd(\mathcal{P}) \text{ and } \neg\mathbf{O}(p) \in frm(\mathcal{P})\}.$$

This proposition allows to construct $\mathcal{P}^{\text{alg}}$ using only syntactical checks, i.e., no need to use an SDL oracle. This is possible because the interaction between deontic formulas in this restricted language is limited and can be described using the rules in the above definition. The first four lines are related to the so-called explosion principle: from a contradiction everything follows. The last two lines are related with the connection between obligation and permission: if it is obligatory then it is permitted. Moreover, it is interesting to note that the maximum number of rules added to $\mathcal{P}^{\text{alg}}$ is $\frac{(\#hd(\mathcal{P}))^2+(2k+1)\times\#hd(\mathcal{P}))}{2}$, where $k$ is the maximum number of formulas in a rule. Therefore, the number of rules is at most quadratic in the number of rules of $\mathcal{P}$.

## Acknowledgments

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

## 1. REFERENCES

[1] A. Artosi, P. Cattabriga, and G. Governatori. Ked: A deontic theorem prover. In *Workshop on Legal Appl. of Logic Programming*, pages 60–76. IDG, 1994.

[2] B. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[3] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *clasp* : A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

[4] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

[5] R. Gonçalves and J. J. Alferes. An embedding of input-output logic in deontic logic programs. In *Deontic Logic in Computer Science, 11th International Conference, DEON 2012, Bergen, Norway, July 16-18, 2012. Proceedings, To appear*, 2012.

[6] R. Gonçalves and J. J. Alferes. Specifying and reasoning about normative systems in deontic logic programming. In W. van der Hoek, L. Padgham, V. Conitzer, and M. Winikoff, editors, *AAMAS*, pages 1423–1424. IFAAMAS, 2012.

[7] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1984.

[8] D. Makinson and L. van der Torre. Constraints for input/output logics. *Journal of Philosophical Logic*, 30:155–185, 2001.

[9] G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.