A Petri net-based notation for normative modeling: evaluation on deontic paradoxes

Giovanni Sileno[2,1], Alexander Boer[1], and Tom van Engers[1]

[1] University of Amsterdam, Leibniz Center for Law, the Netherlands
[2] LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France
g.sileno@uva.nl

Abstract. Developing systems operating in alignment with norms is not a straightforward endeavour. Part of the problems derive from the suggestion that law concerns a system of norms, which, in abstract, in a fixed point in time, could be approached and expressed atemporally, but, when it is contextualized and applied, it naturally deals with a continuous flow of events that modifies the normative directives as well. The paper presents an alternative approach to some of these problems, exemplified by well-known deontic puzzles, by extending the Petri net notation, most common in process modeling, to Logic Programming Petri Nets. The resulting visual formalism represents in a integrated, yet distinct fashion, procedural and declarative aspects of the system under study.

Keywords: Contrary-to-duty, Deontic puzzles, Logic Programming, Petri Nets, Normative Modeling

Introduction

Puzzles and paradoxes are tools for bringing conceptualizations to their boundary conditions, and are therefore relevant to testing formal notations. The deontic logic community has devoted special attention to *paradoxes* constructed with *contrary-to-duty* (CTD) structures (see e.g. [4, 22, 17, 3, 9, 11]). These are "paradoxes" because, although the normative statements look plausible in the natural language form, when each sentence is formalized in *standard deontic logic* (SDL)—the *paper tiger* of normative modeling—either the set of formulas is inconsistent, or one of the formulas is a logical consequence of another formula (see e.g. [2]).

Definition 1 (Contrary to Duty). *A contrary-to-duty (CTD) structure is a situation in which a* primary obligation *exists, and with its violation, a* secondary obligation *comes into existence.*

The importance of CTDs lies in more than just their theoretical aspects: CTDs are fundamental to normative modeling, because they are at the base of *compensatory norms*, prototypical in e.g. contracts. The problem carries definite applicative concerns. This intuitively simple structure produces complex

structures of obligations, prohibitions or permissions applying to sequences of violations or satisfactions relative to the conduct of agents in a regulated social system. Moreover, the secondary obligation may directly contradict the primary obligation. In this case, the conflict *has* to be solved to decide a course of action (i.e. "I am obliged, but I am forbidden, so what should I do?"). This consideration highlights the problem of specifying and treating preferences between idealities, or, in more agentive terms, priorities between commitments.

In this work, we focus on testing CTD structures on Logic Programming Petri Nets (LPPN).[3] This modeling notation has been introduced in [25] with the purpose of integrating and distinguishing in the same visual representation *declarative* aspects (concerning terminology, ontological constraints, normative directives, etc.) and *procedural* aspects (mechanisms, processes, courses of actions, etc.) of the reference system. The resulting common representational ground is proposed as a basis to support a continuous re-alignment in administrative organizations of representations of *law* (norms), of *implementations of law* (services as business processes), and of *action* (behavioural scripts, possibly intentionally characterized). The notation aims therefore to cover a wider class of models (business processes embedded with normative positions, representation of scenarios issued from narratives, agent scripts, etc.) than what usually studied by deontic logic. Furthermore, beside its visual power (in principle increasing its accessibility), it enjoys computational properties as distributed computation, i.e. it does not require necessarily the reference to a global state.[4]

The connection of normative modeling with Petri nets is not completely new; see e.g. [19, 18], and more recently [23]; however, these works mainly focus on events and factual conditions, overlooking normative characterizations. Other works, such as [24], have proposed using Petri nets to formalize contracts; with respect to that work, the present proposal is more specific, as it focuses on minimal CTDs (through the lens of deontic puzzles), but also more general, as it considers the inclusion of declarative bindings in the model.

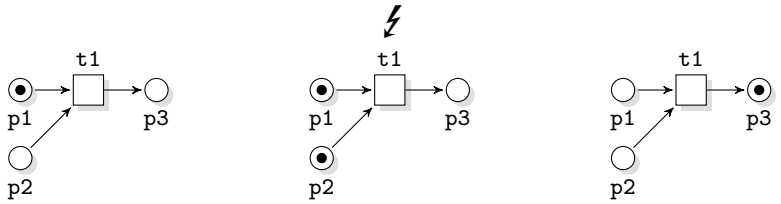
The paper is organized as follows. In § 1, we will present *Logic Programming Petri Nets* (LPPN), delineating the notation and introducing informally a simplified version of its semantics. In § 2, we will review a series of examples, all, save the first, copied from the literature. For each of them, we will propose models of the corresponding scenarios, and attempt to clarify part of the issues encountered in SDL. Discussion and further developments end the paper. A formalization of the propositional version of LPPN can be found in the appendix.

1 Logic Programming Petri Nets

Petri nets are a simple, yet effective computational modeling representation featuring an intuitive visualization (see Fig. 1). They consist in directed, bipartite

[3] Prototypes of LPPN interpreters are available on `http://github.com/s1l3n0/pypneu` and `http://github.com/s1l3n0/lppneu`.

[4] Cf. the recent extension to standard deontic logic by Gabbay and Straßer [6] integrating reactive constructs, an approach in many aspects dual to the present proposal.



(a) not enabled transition, before firing (b) enabled transition and firing (c) the transition has fired

Fig. 1: Example of a Petri net and of its execution (but also of a LPPN *procedural component* when labels are propositions).

graphs with two types of nodes: *places* (visually represented with circles) and *transitions* (with boxes). A place can be connected only to transitions and vice-versa. One or more *tokens* (dots) can reside in each place. The execution of Petri nets is also named "token game": transitions *fire* by consuming tokens from their input places and producing tokens in their output places.[5]

Despite their widespread use in computer science, electronics, business process modeling and biology, Petri nets are generally considered not to be enough expressive for reasoning purposes; in effect, they do not refer explicitly to any informational or representational concept. In their simplest form, tokens are indistinct, and do not transport any data. Nevertheless, usually modelers introduce labels to set up a correspondence between the *modeling* entities and the *modeled* entities. This practice enables them to read the results of a model execution in reference to the modeled system, and therefore it becomes *functional to the use* of the notation, although it is not a requirement for the execution in itself. Further interaction is possible if these labels are processed according an additional formalism, as for instance with the *Coloured Petri Net* (CPN) notation [13], which, for many aspects, is a descendant of *Predicate/Transition Nets* [7]. If its expressiveness and wide application provide reasons for its adoption, the CPN notation introduces many details which are unimportant in our setting (e.g. expressions on arcs); more importantly, it still misses the requirement of processing declarative bindings, necessary, for instance, to model terminological relationships. We opted therefore for an alternative notation.

Whereas Petri nets specify *procedural* mechanisms, LPPNs extend those (a) with Prolog-like *literals* as labels, attached on places and transitions; (b) with nodes specifying (logic) *declarative bindings* on places and on transitions. The notation builds upon the intuition that places and transitions mirror the commonsense distinction between *objects* and *events* (e.g. [1]), roughly reflecting the use of *noun/verb* categories in language [14]: the procedural components can be used to model *transient* aspects of the system in focus; the declarative components to model *steady state* aspects, i.e. those on which the transient is irrelevant or does not make sense (e.g. terminology, ontological constraints, etc.).

[5] For an overview on the general properties of Petri nets see e.g. [21].

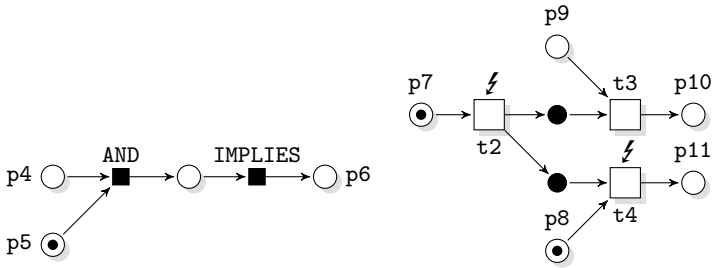


Fig. 2: Examples of LPPN *declarative components*: (a) defined on places, corresponding to the Prolog/ASP code: `p6 :- p4, p5. p5.` (b) defined on transitions, instantaneously propagating the firing where possible (the `IMPLIES` label on black circles is left implicit).

In this paper, for simplicity, we will consider only propositional labeling; with this assumption, the execution model of the LPPN procedural component is the same of *Condition/Event* nets, i.e. Petri nets whose places are not allowed to contain more than one token. For this reason, the Petri net in Fig. 1 can be interpreted as an example of LPPN specifying a procedural mechanism. However, the LPPN notation introduces also logic operator nodes (or *l-nodes*), which apply on places or on transitions. An example of a sub-net with l-nodes for places (small black squares) is given in Fig. 2a. These are used to create logic compositions of places (via operators as `NEG`, `AND`, `OR`, etc). or to specify logic inter-dependencies (via the logic conditional `IMPLIES`). Similarly, transitions may be connected declaratively via l-nodes for transitions (black circles) as in Fig. 2b. These connections may be interpreted as channels enabling *instantaneous propagation* of firing. In this case, it is not relevant to introduce operators as `AND`, for the interleaving semantics, only one source transition may fire per step. To simplify the visual burden, we might left the `IMPLIES` label implicit, exploiting the sense of the arrow to specify the direction of the relation. Operationally, the declarative components are treated integrating the *stable model semantics* used in *answer set programming* (ASP) [15]. This was a natural choice because process execution exhibits a prototypical 'forward' nature, and ASP can be interpreted as providing forward chaining. A formalization of propositional LPPNs can be found in the Appendix.

2 Deontic exercises

2.1 Crossing or not crossing?

Let us start from this minimal, conflicting CTD structure:

You are forbidden to cross the road.
If you are crossing the road, (you have to) cross the road!

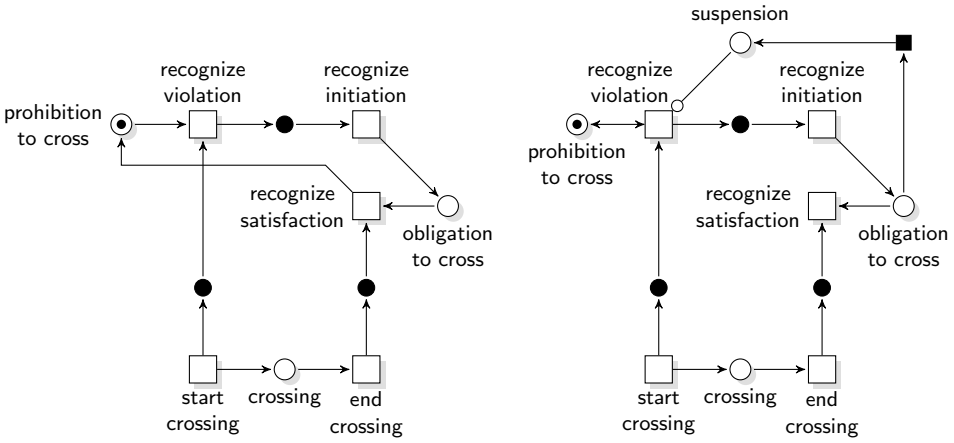


Fig. 3: A minimal, conflicting and event-based *contrary-to-duty* (CTD) structure: a secondary obligation is created after the violation of a primary obligation, and in conflict with it. The CTD may be interpreted: (a) as an *exception*; or (b) as *overriding* the primary obligation. The second is the most accepted option.

This rule of conduct is perfectly plausible: most parents say something similar to their children at some moment. However, its translation in basic deontic logic is not direct. The text suggests, in effect, an underlying model in terms of action: a state-based interpretation would miss the implicit initiation/termination events that make the action-wise prescription sound, and namely:

> *You are forbidden to cross the road.*
> *If you have* started *to cross the road, you are obliged to* finish *crossing.*

This *transitional* aspect can be easily mapped on a LPPN, separating the experiential world from the institutional world, with the second synchronized to the first via *constituting links* determining what *counts as* a violation or a satisfaction (cf. [27]).[6]

In principle, two modeling options are available in regard to the secondary obligation; it can interpreted:

- as an *exception*, thus temporarily retracting the primary obligation (Fig. 3a);
- as *overriding* the primary obligation, which persists concurrently (Fig. 3b).

The second option requires an additional treatment, because it brings two contrary opposite positions to hold concurrently. Similarly to what suggested in the

[6] With respect to constitutive rules, the LPPN notation enables to easily distinguish *classificatory constitutive rules* (e.g. "a bike counts as a vehicle") from *constitutive event rules* (e.g. "raising a hand counts as making a bid"), as they are modeled respectively using black boxes or black circles. Most formalizations of constitutive rules consider only on the first aspect (e.g. [10]), cf. the overview in [27].

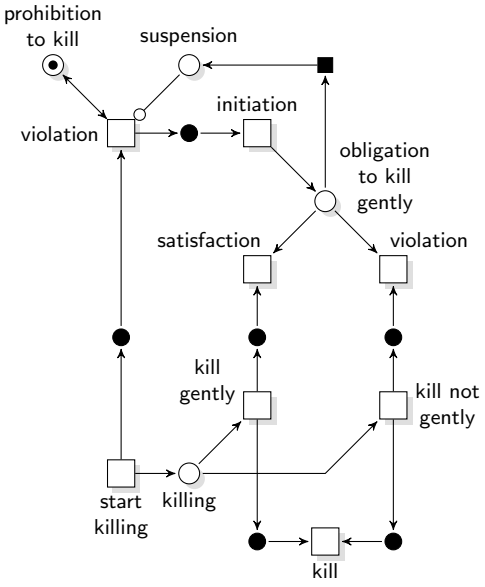


Fig. 4: *Gentle murderer* case.

literature, this can be solved introducing an explicit *ordering* between positions, which depends on how close to *ideal* is the world/context they are referring to (see e.g. [16, 22]). In the proposed Petri net an aspect of this mechanism—the fact that the secondary obligation is put in force *in response to the violation* of the primary one—is already reified in the topology. To capture the remaining part, i.e. that the second is *contextually overriding* the first, we need to order them in the opposite sense: the last obligation created is the one with most priority and should be the only active, *suspending* the previous ones. This can be done introducing an *inhibiting arc* (visualized in Fig. 3b as an arrow with a circle-shaped head).[7] The resulting design can be seen as a model of *salience*.

2.2 Gentle murderer

The previous CTD model gives us the basic instruments to proceed. Let us start from the classic case of the "gentle murderer", given by Forrester [5]:

> *It is forbidden to kill,*
> *but if one kills, one ought to kill gently.*

This example is very similar to the previous one, except that the target of the secondary obligation is subsumed by the target of the first one. Because our notation explicitly accounts for a declarative dimension for events, we can directly map this relation (Fig. 4).

[7] Inhibiting arcs goes from places to transitions. If the input place of an inhibiting arc is occupied, its output transition is disabled.

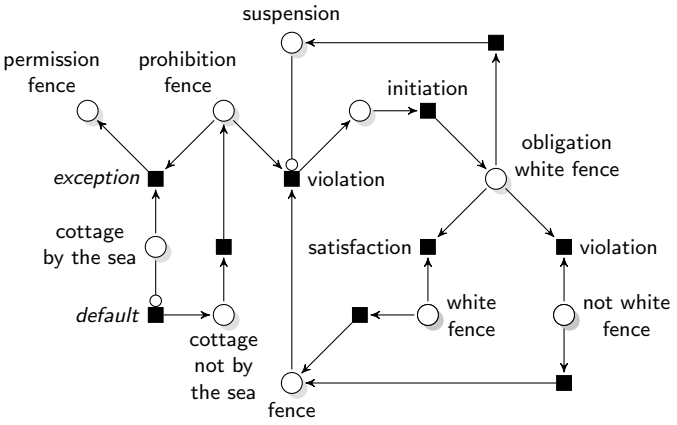


Fig. 5: *White fence* case.

2.3 White fence

Now, we consider a static and extended variation proposed by Prakken and Sergot [22], the "white fence" case:

> *There must be no fence.*
> *If there is a fence, it must be a white fence.*
> *If the cottage is by the sea, there may be a fence.*

This example shows the importance of distinguishing exceptions from overriding effects due to CTDs (and therefore supports the second interpretation of CTD given in § 2.1). In principle, a rule specifies a CTD if its premise is the negation of the target of the obligation in the consequent of another rule. A rule specifies an exception if it has as consequent the negation of the consequent of another rule, *and* it has a lower *priority* than the first one (exceptions are by definition subordinate to some *normal* conditions). In effect, the two rules can be read as referring to a *priority-based representation* [26]. Considering part of the "white fence" case in propositional form, we have:

$$Forb(fence)$$
$$sea \rightarrow Perm(fence)$$

which can be translated to the corresponding *constraint-based* representation:

$$\neg sea \rightarrow Forb(fence)$$
$$sea \rightarrow Perm(fence)$$

This treatment gives a hint as to how to deal with *exceptions*—that is, it helps make explicit an enchaining of negations of the premises following the inverse ordering of salience. The fastest solution to avoiding conflicts in the case of belief

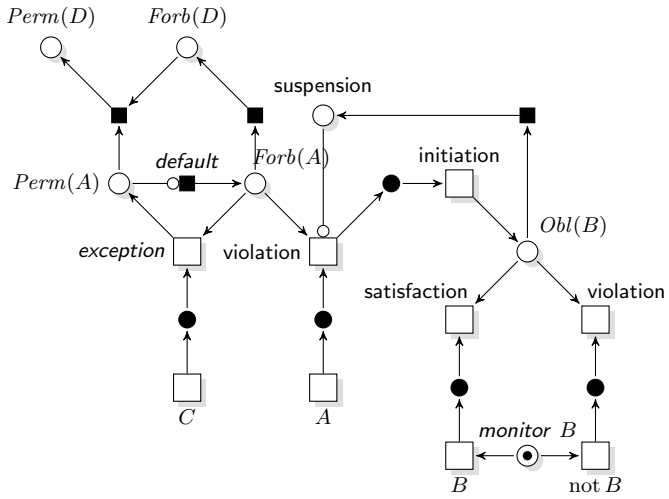


Fig. 6: *Privacy act* case.

revision is to not reify directly the *default* position (in this case, prohibition against having a fence), but to generate it through a *default rule* [26]:

$$\text{not } sea \rightarrow Forb(fence)$$

The resulting model is illustrated in Fig. 5.

2.4 Privacy act

Recently, Governatori [8] has proposed the case of a Privacy Act (fictional, but based on actual Australian normative provisions):

i. *The collection of personal information is forbidden, unless acting on a court order authorising it.*
ii. *The destruction of illegally collected personal information before accessing it is a defence against the illegal collection of the personal information.*
iii. *The collection of medical information is forbidden, unless the entity collecting the medical information is permitted to collect personal information.*

The following deontic interpretation is proposed:

i. Forbidden A. If C, then Permitted A.
ii. If Forbidden A and A, then Obligatory B.
iii. Forbidden D. If Permitted A, then Permitted D.

A, B, C, and D in this specific case are actions; (ii) specifies a CTD, (i) and (iii) provide rules based on a priority-based representation. As before, we extract explicitly the defaults. The negation of permission of A in (iii) can be interpreted as the prohibition of A, thus converging to the default in (i). For completeness, we have reported in Fig. 6 the monitoring place from which the event B or not B is recognized.

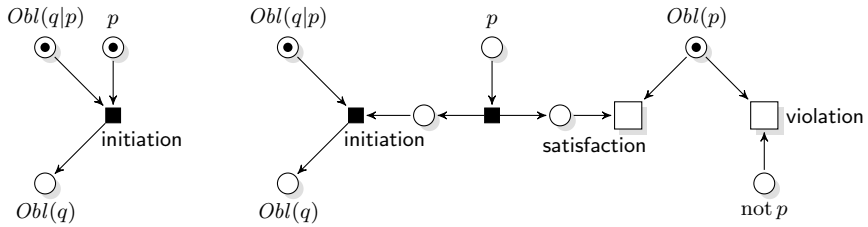


Fig. 7: Factual and deontic detachments.

2.5 Detachment principles

In the deontic logic literature, two types of "detachment principles" are recognized as relevant. The first is called *factual detachment* (FD):

$$p \wedge Obl(q|p) \rightarrow Obl(q) \tag{1}$$

The second is known as *deontic detachment* (DD):

$$Obl(p) \wedge Obl(q|p) \rightarrow Obl(q) \tag{2}$$

In our framework, a conditional directive or commitment is seen as a *susceptibility* to a condition that *creates* or *implies* the directive depending upon whether the connective is a *causal* or *logical dependence*. The two principles can be then translated using the LPPN notation. Focusing on the logical dependence case, the result is seen in Fig. 7. The pictures show that the first principle is satisfied by the notation semantics; on the contrary, the second principle, which is based on an *anticipation* of the normal conditional, is not satisfied.

2.6 Derived obligation

Consider these two sentences:

- *Bob's promise to meet you commits him to meeting you.*
- *It is obligatory that if Bob promises to meet you, he does so.*

Although in natural language their difference is arguable, in the literature they have been formalized using two distinct deontic formulations:

- $p \rightarrow Obl(m)$
- $Obl(p \rightarrow m)$

What is the difference between the two formulations in our framework? The second formula can be translated with the consideration that the obligation of *something* consists of two recognition rules about satisfaction and violation, by default anchored respectively to *something* and to $\neg$*something*. As an object, the

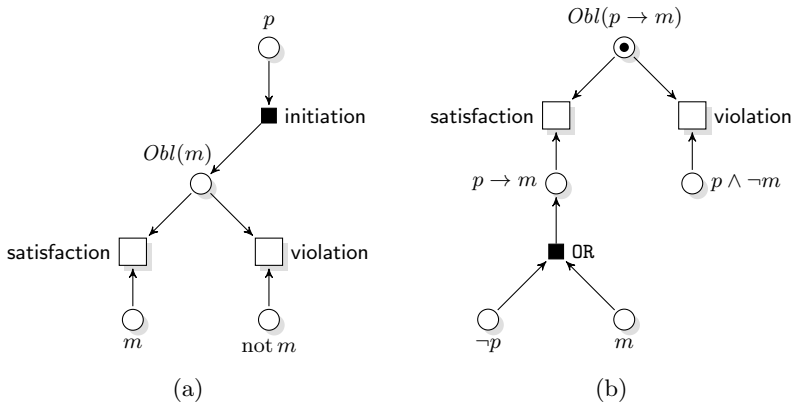


Fig. 8: Derived obligation modeled as $p \rightarrow Obl(m)$ or as $Obl(p \rightarrow m)$.

conditional within the obligation can be transformed using the *material implication*.[8] The result are reported in Fig. 8. As we see in the picture, both models are violated in the same situation (p and $\neg m$); however, the second includes two satisfied situations not accounted for in the first ($\neg p$). In other words, the first derived obligation precisely discriminates the elements producing the violation. The second takes an explicit position also on the satisfying elements.

2.7 Chisholm's paradox

At this point, we can finally model the "paradox" proposed by Chisholm [4]:

> *It ought to be that Jones goes (to the assistance of his neighbors).*
> *It ought to be that if Jones goes, then he tells them he is coming.*
> *If Jones doesn't go, then he ought not tell them he is coming.*
> *Jones doesn't go.*

This was seen as a paradox, because if we model it as:

i. $Obl(go)$
ii. $Obl(go \rightarrow tell)$
iii. $\neg go \rightarrow Forb(tell)$
iv. $\neg go$

and we apply both deontic and factual detachments, we find an inconsistency. More precisely, from (i) and (ii), using of deontic detachment, we derive $Obl(tell)$,

[8] The specific example from which we started is not based on a logic conditional, but on a causal connective, at least in the case of "if Bob promises to meet you, then he does so". In this case, the use of material implication is not a perfect fit, as the temporal shift between the promise and the meeting falsifies the derived constraint, at least on a transient basis. On a steady state analysis, however, this simplification may be applied.

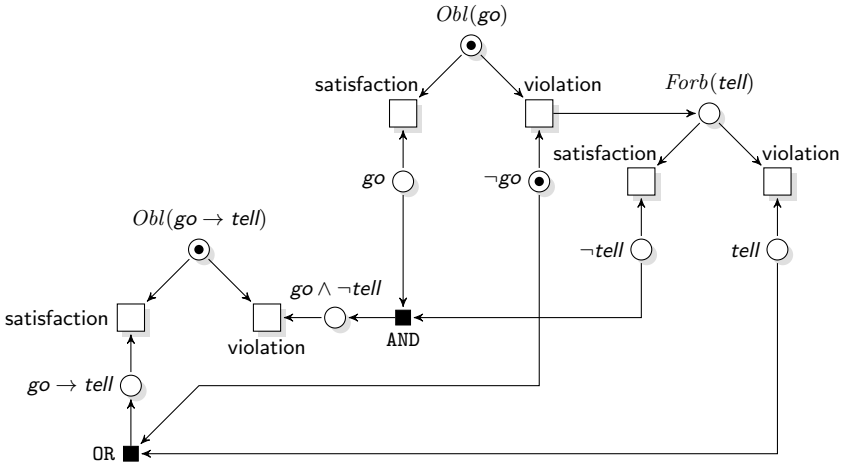


Fig. 9: *Chisholm's paradox* case.

while from (iii) and (iv), using factual detachment, we derive $Obl(\neg tell)$. However, representing this model using our notation as in Fig. 9, we do not find any specific issue. The fact $\neg go$ satisfies $Obl(go \rightarrow tell)$, but violates $Obl(go)$, and for this reason, $Forb(tell)$ is instantiated. Depending on whether *tell* becomes true, this may or may not be violated.

2.8 Residential neighbourhood

As we observed above, the difference in natural language between the two types of derived obligations is arguable. In order to show what would happen if we interpret the two as conditional obligations, we consider the non-agent version of the Chisholm's paradox proposed by Prakken and Sergot [22]:

> *There must be no dog.*
> *If there is no dog, there must be no warning sign.*
> *If there is a dog, there must be a warning sign.*
> *There is a dog.*

Written in this way, the modeling is straightforward (Fig. 10).

3 Discussion and further developments

At superficial level, the paper presented examples of application of a notation introduced for wider modeling purposes [25]. This served as an important exercise to (partially) evaluate its practical functionality, as one may capture the subtle problems related to modeling in a certain domain (in this case normative modeling) only by approaching the specific issues that have been raised by practitionners and scholars in that domain. The introduction of *defaults*, *exceptions*,

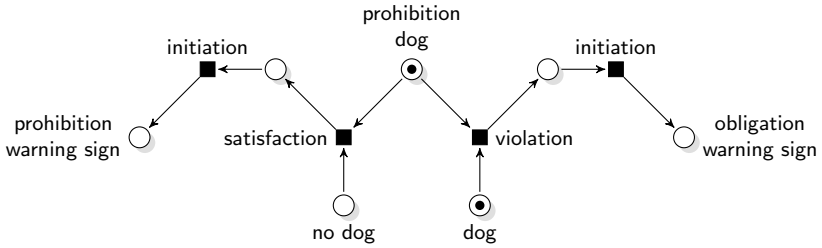


Fig. 10: *Residential neighbourhood* case.

and *suspensions* in the notation presented here is a preliminary proposal. Some of these aspects have been treated more in more detail elsewhere (e.g. *defaults* in priority-based rule-bases [26], proceeding along [12]), while others require further investigation (e.g. *suspension*, cf. [20]). Nevertheless, the exercise yields concrete practical and theoretical results.

From a practical point of view, it makes a case supporting normative modeling with notations similar to those used in business process modeling, thus potentially facilitating cross-fertilization between theoretical to operational settings. From a theoretical point of view, we observed for instance that our conceptual framework does not entail the *deontic detachment* principle, hinting at a more general *minimal commitment* taken by the notation. In other words, i.e. the notation does not provide any rule *a priori* to conclude whether $Obl(A) \wedge Obl(B)$ is the same as $Obl(A \wedge B)$. This neutral starting point can be used to evaluate the alternative impact of different axioms proposed in the literature, in affinity with approaches like *input/output logic* [17].

More importantly, while working on these exercises, we appreciated the crucial interplay between static and dynamic aspects (one of the issues underlying many deontic puzzles). The LPPN notation, requiring the explicitation of procedural and declarative aspects, highly facilitated this task, but our exercise suggests further research on the modeling methodology. For instance, in Chisholm's paradox, we considered *go* and *tell* as labels of places, but strictly speaking, they should be attached to transitions (as in the *privacy act* case). With this choice, we would require making explicit the *occurrence* of the events as places, in order to evaluate the material implication. Does this simplification hint at a more general pattern? We modeled the nodes concerning violation and satisfaction as l-nodes only when their transformational nature was certain. In general, however, they may be *transformational* (when they simply identify whether a violation or satisfaction holds at the moment) or *reactive* (when they reify the fact that a violation occurred in that moment, or when they cause any change on the inputs, e.g. removing the obligation). Here we notice again the interplay between static and dynamic aspects. In the future, an investigation of boundary cases may help in formulating a more general theory on how to decide upon the level of abstraction, and about whether it is possible to identify or elaborate general patterns, that, depending on circumstances, are read in one form or in the other.

References

1. Breuker, J., Hoekstra, R.: Core concepts of law: taking common-sense seriously. Proc. of Formal Ontologies in Information (2004)
2. Broersen, J., van der Torre, L.: Ten Problems of Deontic Logic and Normative Reasoning in Computer Science. Lectures on Logic and Computation, 55–88 (2012)
3. Carmo, J., Jones, A.: Deontic logic and contrary-to-duties. Handbook of philosophical logic 8, 265–343 (2002)
4. Chisholm, R.M.: Contrary-to-duty imperatives and deontic logic. Analysis 24(2), 33–36 (1963)
5. Forrester, J.W.: Gentle Murder, or the Adverbial Samaritan. The Journal of Philosophy 81(4), 193–197 (1984)
6. Gabbay, D.M., Straßer, C.: Reactive standard deontic logic. Journal of Logic and Computation 25(1), 117–157 (2015)
7. Genrich, H.J.: Predicate/Transition Nets. In: Proc. Advances in Petri nets 1986. 207–247 (1987)
8. Governatori, G.: Thou Shalt is not You Will. Tech. rep., NICTA (2015)
9. Governatori, G., Rotolo, A.: Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. Australasian Journal of Logic 4, 193–215 (2006)
10. Grossi, D., Meyer, J.J.C., Dignum, F.: Classificatory Aspects of Counts-as: An Analysis in Modal Logic. Journal of Logic and Computation 16(5), 613–643 (2006)
11. Hansen, J., Pigozzi, G., Van Der Torre, L.: Ten philosophical problems in deontic logic. Normative Multi-agent (2007)
12. Horty, J.F.: Rules and Reasons in the Theory of Precedent. Legal Theory 17(01), (2011)
13. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag (1996)
14. Kemmerer, D., Eggleston, A.: Nouns and verbs in the brain: Implications of linguistic typology for cognitive neuroscience. Lingua 120(12), 2686–2690 (2010)
15. Lifschitz, V.: What Is Answer Set Programming? Proc. of 23rd the AAAI Conf. on AI (2008)
16. Makinson, D.: Five faces of minimality. Studia Logica 339–379 (1993)
17. Makinson, D., Van Der Torre, L.: Input/output logics. Journal of Philosophical Logic (2000)
18. Meldman, J., Fox, S.: Concise Petri Nets and Their Use in Modeling the Social Work (Scotland) Act 1968. Emory Law Journal 30, 583–630 (1981)
19. Meldman, J., Holt, A.: Petri nets and legal systems. Jurimetrics journal 12(2), 65–75 (1971)
20. Meneguzzi, F., Telang, P., Singh, M.: A first-order formalization of commitments and goals for planning. In: Proc. of the 27th AAAI Conf. on AI. 697–703 (2013)
21. Murata, T.: Petri nets: Properties, analysis and applications. Proc. of the IEEE 77(4) (1989)
22. Prakken, H., Sergot, M.: Contrary-to-duty obligations. Studia Logica 57(1), 91–115 (jul 1996)
23. Purvis, M.A.: Dynamic Modelling of Legal Processes with Petri Nets. Ph.D. thesis, University of Otago (1998)
24. Raskin, J.F., Tan, Y.H., van der Torre, L.: How to model normative behavior in Petri nets. Proc. of the 2nd Workshop on Formal Models of Agents 223–241 (1996)
25. Sileno, G.: Aligning Law and Action. Ph.D. thesis, University of Amsterdam (2016)

26. Sileno, G., Boer, A., van Engers, T.: A Constructivist Approach to Rule Bases. In: Proc. of the 7th Int. Conf. on Agents and AI (ICAART 2015). (2015)
27. Sileno, G., Boer, A., van Engers, T.: Revisiting Constitutive Rules. In: Proc. of the 6th Workshop on AI and the Complexity of Legal Systems (AICOL 2015) (2015)

A Formalization

Here we present a simplified version of the LPPN notation considering only a *propositional* labeling. We start from the definition of propositional literals derived from ASP [15], accounting for strong and default negation.

Definition 2 (Literal and Extended literals). *Given a set of propositional atoms A, the set of* literals $L = L^+ \cup L^-$ *consists of* positive literals *(atoms)* $L^+ = A$, negative literals *(negated atoms)* $L^- = \{-a \mid a \in A\}$, *where '$-$' stands for* strong negation.[9] *The set of* extended literals $L^* = L \cup L^{\text{not}}$ *consists of literals and* default negation literals $L^{\text{not}} = \{\text{not}\, l |\ l \in L\}$, *where 'not' stands for* default negation.[10]

We denote the basic topology of a Petri net as a procedural net.

Definition 3 (Procedural net). *A* procedural net *is a* bipartite directed graph *connecting two finite sets of nodes, called* places *and* transitions. *It can be written as $N = \langle P, T, E \rangle$, where $P = \{p_1, \ldots, p_n\}$ is the set of* place *nodes; $T = \{t_1, \ldots, t_m\}$ is the set of* transition *nodes; $E = E^+ \cup E^-$ is the set of* arcs *connecting them: E^+ from transitions to places, E^- from places to transitions.*

LPPNs consists of three components: a procedural net specifying causal or temporal relationships, and two declarative nets specifying respectively logical dependencies at the level of objects or ongoing events (on places), and on impulse events (on transitions). Furthermore, *propositional* LPPNs build upon a boolean marking on places (like *condition/event* nets).

Definition 4 (Propositional Logic Programming Petri Net). *A* propositional Logic Programming Petri Net $LPPN_{\text{prop}}$ *is a Petri Net whose places and transitions are labeled with literals, enriched with declarative nets of places and of transitions. It is defined by the following components:*

- $\langle P, T, PE \rangle$ *is a procedural net; PE stands for* procedural edges*;*
- $C_P : P \to L^*$ *and* $C_T : T \to L$ *are labeling functions, associating literals respectively to places and to transitions;*
- $OP = \{\neg, -, \wedge, \vee, \to, \leftrightarrow, \ldots\}$ *is a set of logic operators.*
- LP *and* LT *are sets of* logic operator nodes *(in the following called* l-nodes*) respectively for places and for transitions.*

[9] Strong negation is used to reify an explicitly false situation (e.g. "It does not rain").
[10] Default negation is used to reify a situation in which something cannot be retrieved/inferred (e.g. 'It is unknown whether it rains or not').

- $C_{LP} : LP \to OP$ *maps each l-node for places to a logic operator; similarly,* $C_{LT} : LT \to OP$ *does the same for l-nodes for transitions.*
- $DE_{LP} = DE^{+}_{LP} \cup DE^{-}_{LP}$ *is the set of arcs connecting l-nodes for places to places; similarly,* $DE_{LT} = DE^{+}_{LT} \cup DE^{-}_{LT}$ *for l-nodes for transitions and transitions.*[11]
- $M : P \to \{0, 1\}$ *returns the marking of a place, i.e. whether the place contains (1) or does not contain (0) a token.*

Note that if $LP \cup LT = \varnothing$, we have a *strictly procedural* $LPPN_{\text{prop}}$, i.e. a standard binary Petri net. If $T = \varnothing$, we have a *strictly declarative* $LPPN_{\text{prop}}$, that can be directly mapped to an ASP program.

With respect to the *operational semantics*, the execution cycle of a LPPN consists of four steps: (1) given a "source" marking M, the bindings of the declarative net of places entail a "ground" marking M^*; (2) an enabled transition is selected to *pre-fire*, so determining a "source" *transition-event* e; (3) the bindings of the declarative net of transitions entail all propagations of this event, obtaining a set of *transition-events*, also denoted as the "ground" *event-marking* E^*; (4) all transition-events are fired, producing and consuming the relative tokens. The steps (1) and (3) are processed by an ASP solver: the declarative net of places (respectively transitions) is translated as *rules*, tokens (transition-events) are reified as *facts*; the ASP solver takes as input the resulting program and, if satisfiable, it provides as output one or more ground marking (one or more sets transition-events to be fired). For the steps (2) and (4), the operational semantics distinguishes the *external* firings (started by the execution) from the *internal* firing, immediately propagated (triggered by the declarative net of transitions).

Definition 5 (Enabled transition). *A transition* t *is* enabled *in a ground marking* M^* *if a token is available for each input places:*

$$Enabled(t) \equiv \forall p_i \in \bullet t, M^*(p) = 1$$

Similarly to what marking is for places, we consider an *event-marking* for transitions $E : T \to \{0, 1\}$. $E(t) = 1$ if the transition t produces a transition-event e. Each step s has a "source" event-marking E.

Definition 6 (Pre-firing). *An enabled transition* t pre-fires *at a step* s *if it selected to produce a* transition-event*:*

$$\forall t \in Enabled(T) : t \textit{ pre-fires} \equiv E(t) = 1$$

As we apply an *interleaving semantics* for the pre-firing, the interpreter selects only one transition to pre-fire per step; for any other t', $E(t') = 0$.

Definition 7 (Firing). *An enabled transition* t fires *by propagation, consuming a token from each input place, and forging a token in each output place:*

$$\forall t \in Enabled(T) : t \textit{ fires} \equiv$$
$$E^*(t) = 1 \leftrightarrow \forall p_i \in \bullet t : M'(p_i) = 0 \ \wedge \ \forall p_o \in t\bullet : M'(p_o) = 1$$

[11] Note that $DE^{-}_{LT} \subseteq (T \cup P) \times LT$, i.e. these edges go from transitions *and* places (modeling contextual conditions) to l-nodes for transitions.