Towards the Norm-Aware Agent: Bridging the Gap Between Deontic Specifications and Practical Mechanisms for Norm Monitoring and Norm-Aware Planning

Sofia Panagiotidi(✉), Sergio Alvarez-Napagao, and Javier Vázquez-Salceda

Universitat Politècnica de Catalunya-BarcelonaTECH, Edifici Omega, Despatx 206-207 C/ Jordi Girona Salgado 1-3, 08034 Barcelona, Spain
{panagiotidi,salvarez,jvazquez}@lsi.upc.edu

Abstract. In the agents' literature, norms have been studied from multiple perspectives, but while formalisations tend to be disconnected from possible implementations due to the lack of differentiation between abstract norm and norm instantiation, on the other hand implementations tend to be weak groundings of deontic logics, tightly coupled to one particular implementation domain. Furthermore, different formalisations are typically used for norm enforcement and norm reasoning. In this paper we report on our attempt to bridge this gap by reducing from deontic statements to structural operational semantics (for norm monitoring) and to planning control rules (for practical normative reasoning). We hint at the feasibility of the translation of these semantics to actual implementation languages (Clojure and Drools for norm monitoring and TLPlan for norm-aware planning). Finally we discuss the limitations of our approach and suggest some improvements and future lines of research.

Keywords: Deontic logics · Normative systems · Planning · Monitoring

1 Introduction

In literature the concept of *norms* has been defined from several perspectives [1]: as a rule or standard of behaviour shared by members of a social group, as an authoritative rule or standard by which something is judged, approved or disapproved, as standards of right and wrong, beauty and ugliness, and truth and falsehood, or even as a model of what should exist or be followed, or an average of what currently does exist in some context. Moreover, from the Artificial Intelligence community there has been a continuous effort on researching how to formalise norms from a logic perspective, on one hand, and how to make them feasibly computable, on the other hand.

In this work we will focus on the regulative aspects of sets of norms (that we will call normative specifications), seen as a way to model the governance of

T. Balke et al. (Eds.): COIN 2013, LNAI 8386, pp. 346–363, 2014.
DOI: 10.1007/978-3-319-07314-9_19,

distributed, agent-oriented systems by explicitly specifying the agents' expected behaviour.

The main advantage of normative specifications over other governance mechanisms is that norms make explicit the (social) expectation about *what* is expected to happen, but not *how* the agents are supposed to bring it about. Therefore normative specifications allow the design of complex social setups while giving enough flexibility to give agents some level of autonomy to, e.g., react to unexpected states of the system.

In literature there is a lot of work on normative systems' formalisation (mainly focused in Deontic-like formalisms [2]) which is declarative in nature, focused on the expressiveness of the norms [3], the definition of formal semantics [4–7] and the verification of consistency of a given set [8,9]. There are some works that focus on norm compliance and norm monitoring [6,7,9–12] with varying degrees of covered abstraction level and allowed flexibility. Also there is some work on how agents might take norms into account when reasoning [5,13–16], but few practical implementations exist that cover the full BDI cycle, as many approaches do not include the *means-ends reasoning step* (that is, deciding *how* to achieve *what* the agent is aiming for). However, we have found no work in the literature that (1) formally connects the deontic aspects of norms with their operationalisation, (2) properly distinguishes between abstract norms and their (multiple) instantiations at run-time, (3) formalises the operational semantics in a way that ensures flexibility in their translation to actual implementations while ensuring unambiguous interpretations of the norms, and (4) covers both institutional-level norm monitoring and individual agent norm-aware reasoning to ensure that both are aligned.

In this paper, we present a proposal to bridge the gap between a single norm formalisation and the actual mechanisms used for both (rule-based) norm monitoring and norm-aware planning. Taking advantage of a recent trend in the Planning community to use Linear Temporal Logic (LTL) formulas as strong and soft constraints on plan trajectories (e.g., TLPlan [17] and PDDL 3.0 [18]), we have chosen LTL as a bridge from the norm specification to its implementation by reducing deontic-based norm definitions to temporal logic formulae which, in turn, can be translated into both rule-based and planning operational semantics.

The paper is organised as follows: Sect. 2 introduces the formalism of temporal logics to be used as basis in following sections. Section 3 focuses on the concepts of norm instance and norm lifecycle, and discusses how norm operationalisation is usually handled in literature. In Sect. 4 we focus on the semantics of norms and norm instances from the deontic statement level, while in Sect. 5 we focus on the operational semantics and how it can be used in practical implementations for monitoring and planning. Finally, in Sect. 6 we present some conclusions and future lines of work.

2 Linear Temporal Logic

LTL [19] is built up from a finite set of predicates $\mathscr{L}$, the logical operators $\neg$ and $\vee$ (logical operators $\wedge, \rightarrow, \leftrightarrow$, true, and false can be derived by the primitive

ones), and the temporal modal operators **X** (next) and **U** (until). Formally, the set of LTL formulas over $\mathscr{L}$ is inductively defined as follows:

- if $p \in \mathscr{L}$ then p is a LTL formula;
- if ψ and ϕ are LTL formulas then $\neg\psi, \phi \vee \psi, \mathbf{X}\psi$ *and* $\phi\mathbf{U}\psi$ are LTL formulas.

We define a substitution (grounding) $\theta = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, ..., x_i \leftarrow t_i\}$ as the substitution of the terms $t_1, t_2, ..., t_i$ for variables $x_1, x_2, ..., x_i$ in a formula $f \in \mathscr{L}$. Thus, $\theta(f(x_1, x_2, ..., x_i)) \equiv f(t_1, t_2, ..., t_i)$. A *state of the world* s_t is a set of atomic predicates grounded by θ holding true at a specific moment. An LTL model $M = (S, \Re, \theta)$ consists of a non empty set S of states, an accessibility relation $\Re$ (connecting a state to another) and an substitution θ for predicates. A full path π in M is a sequence $\pi =< s_0, s_1, s_2, \ldots >$ such that for every $i \geq 0$, s_i is an element of S and $s_i \Re s_{i+1}$, and if π is finite with s_n its final state, then there is no state s_{n+1} in S such that $s_n \Re s_{n+1}$. Additionally, let π_i be the subpath of π starting from the i'th state of π, i.e. $\pi_i =< s_i, s_{i+1}, \ldots >$. Validity of an LTL formula ϕ on a model $M = (S, \Re, \theta)$ over a path π, written as $M, \pi \models \theta(\phi)$, is defined as:

- $M, \pi \models \theta(p) \quad \Leftrightarrow \theta(p) \in s_0$
- $M, \pi \models \neg\theta(\phi) \quad \Leftrightarrow$ not $M, \pi \models \theta(\phi)$
- $M, \pi \models \theta(\phi) \vee \theta(\psi) \Leftrightarrow M, \pi \models \theta(\phi)$ or $M, \pi \models \theta(\psi)$
- $M, \pi \models \mathbf{X}\theta(\phi) \quad \Leftrightarrow M, \pi_1 \models \theta(\phi)$
- $M, \pi \models \theta(\phi)\mathbf{U}\theta(\psi) \Leftrightarrow \exists n > 0$ such that:
  (1) $M, \pi_n \models \theta(\psi)$ and
  (2) $\forall\, i\, with\ 0\ \leq i < n\ :\ M, \pi_i \models \theta(\phi)$

Additional temporal operators are **G** for always (globally), **F** for eventually (in the future), **R** for release and **W** for weakly until. Details about LTL can be found at [19].

3 Norms and Norm Instances

Searle [20] distinguishes between two types of norms: regulative rules, which describe ideal situations from an institutional perspective in terms of obligations, prohibitions and permissions, and constitutive rules, which allow to construct social reality by expliciting the relationship between brute facts and institutional events. The main difference between both types of norms is that while constitutive rules, by their very nature, are categorical, regulative rules are conditional, in the sense that they specify every applicable condition of each particular norm [4].

Although there have been recent attempts to make regulative rules concrete by the reduction to constitutive rules [21], in general regulative norms based on deontic statements have been the most common way to represent normative constraints in multi-agent systems. In such systems, thus, norms are expressed as computer-readable specifications based on deontic logics.

3.1 Norm Operationalisation and Levels of Abstraction

However, operationalisation of regulative norms[1] is not straightforward. Deontic statements express the existence of norms, rather than the consequences of following (or not following) them [22]. In order to implement agents and institutional frameworks capable of reasoning about norms, we need to complement deontic logics with semantics defining fulfilment and violation – among other operational normative concepts. Examples of work on this direction are abundant and for many different purposes, i.e., compliance [6,7,9,11,12,23], verification [8,10,24,25], or agent behaviour [5,13–16].

While it is true that most of them define semantics to interpret norms, there seems to be a disconnection between such semantics and either (1) the deontic logics they are supposed to be based upon; or (2) the operational level closer to the actual practical implementation. For instance, [9] defines a norm-operationalisation language that can be connected with higher level abstractions, but it is not clear whether it can be translated into generic rule-based languages. [6] presents a rule-based language with constraints, with an implementation on Prolog, on top of which other higher-level languages can be formalised, but with no direct relationship to deontic logics. On this line of work, approaches such as [7,13] define clear operational semantics by the use of syntax loosely inspired by, but not directly related to, deontic statements.

An approach that is close to bridge this gap is presented in [26] by specifying formal methods for the implementation of norm enforcement and the automatic creation of protocols based on constraints specified by the norms. However, this proposal focuses on norm modelling from an institutional point of view, not covering the agent perspective (i.e., how norms influence the agent behaviour). A second limitation is that it does not get to the implementation level and, in fact, there does not seem to be a straightforward way to achieve it. Furthermore, it includes no treatment of the consequences, i.e. norm reparation.

In summary, there are many approaches that tackle different parts of the formalisation of norm operationalisation. One of the purposes of this paper is, thus, to complement these approaches by filling the gaps that exist between the deontic statements and both rule-based and planning operationalisation by means of (1) additional predicates representing norm fulfilment and violation, and (2) an intermediate representation based on temporal logics (see Sect. 4).

3.2 Identification of Norm Instances

A related issue that is somehow missing in general in the literature is a clear separation between an abstract norm and a particular (contextual) instantiation of the norm. This problem was already discussed by Abrahams and Bacon in [27]: *"since propositions about norms are derived from the norms themselves, invalid or misleading inferences will result if we deal merely with the propositions rather*

[1] In the rest of the paper we will use the term *norms* or *regulative norms* to refer to Searle's *regulative rules*.

than with the identified norms[2] *that make those propositions true or false"*. This issue is not banal, as it has implications on the operational level: in order to properly check norm compliance, norm instantiations have to be tracked in an individual manner, case by case.

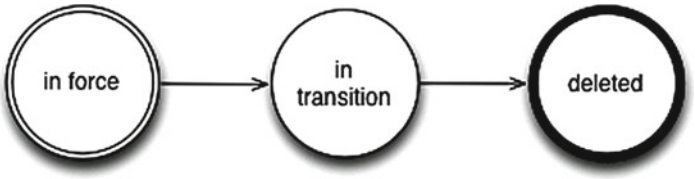

Fig. 1. Norm lifecycle

We find useful, at this point, to stress the fact that the lifecycles of a norm, and of a norm instance, should be differentiated because they are different in essence. The lifecycle of a norm (see Fig. 1) deals with its validity in the normative system: a norm is *in force* when it can be fully activated, monitored, and enforced; *in transition* when it is being removed and cannot be activated anymore, but the effects of past activations have to be tracked until their end; and *deleted* when the history of the norm is to be kept but it can have no further effect on the normative system. Therefore, such lifecycle is related to the concepts of promulgation, abrogation and derogation, out of the scope of this paper. On the other hand, the lifecycle of a norm instance deals with the fulfilment/violation of each particular instance.

The concept of norm instance life-cycle has been treated by different authors, e.g. [7, 27–29], but with no real consensus. Taking those interesting elements that would allow to manage norms with the concepts of activation, maintenance, fulfilment and reparation, a suitable norm life-cycle would be similar to the one based on the automata depicted in Fig. 2. A norm instance gets activated due to a certain activating condition and starts in an (A)ctive state, but if at some point a certain maintenance condition is not fulfilled, the norm instance gets into a (V)iolation state. If the norm instance is (A)ctive and a certain deactivation (or fulfilment) condition is achieved, the norm gets (D)eactivated[3]. Usually reparations are not treated explicitly, but in our proposal we add the concept for completeness. If a norm instance is (V)iolated, fulfilling a reparation condition can bring it back to the (A)ctive state, but if the deactivation condition occurs while violated, only by fulfilling the same reparation condition (VD state) the norm instance can be (D)eactivated. A (V)iolated norm instance could not ever get repaired, so for safety we use a *timeout* condition[4] to make sure the norm instance is not alive forever and thus mark those permanent violations as (F)ailures.

[2] In this paper, we will denote such identified norms as *norm instances*.
[3] Please note that we assume the deactivation condition to eventually happen.
[4] The timeout condition is evaluated as starting at the point of time of violation.

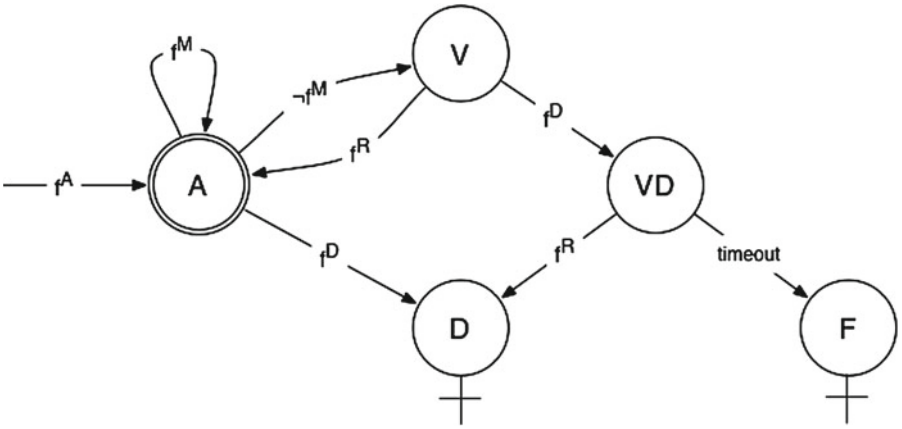

Fig. 2. Norm instance lifecycle with reparation and timeout handling

Once there is a norm life-cycle the question to answer is how to deal with it from an operational perspective. Abrahams and Bacon [27] solve this problem by means of *occurrences* of the predicates contained in the deontic operator, but there are cases in which this can be insufficient, e.g., when the obligation defines a deadline or its instantiation depends on contextual information. More recently, some works have been advancing in the direction of tackling this issue. For example, by treating instantiated deontic statements as first-class objects of a rule-based language [23,30]. However, as these deontic statements are already implicitly identifying the norm instance, there is no explicit tracking of which elements of the domain are involved in fulfilling or violating. Other approaches declare the norm only at the abstract level and the tracking of the norm instance, and implicitly of the norm instance lifecycle, is purely done at the operational level [7,11,12].

4 Formalisation

In this section, we present a proposal for a deontic logic for support for norm instantiation via obligations parametrized by three states (conditions). For the purpose of this formalisation, we assume the use of a predicate based propositional language $\mathscr{L}$ as in Sect. 2. We also adopt the notion of state from the same section.

4.1 Norms

In this paper we define a norm following a modified version of the *abstract norm* definition from [7], adding elements for tracking of reparation of violations:

Definition 1. *We define a norm n as a tuple* $n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout \rangle$, *where:*

- α *is the agent obliged to comply with the norm,*
- f_n^A *is the activating condition of the norm,*
- f_n^M *is the maintenance condition of the norm,*
- f_n^D *is the deactivation condition of the norm,*
- f_n^R *is the repair condition of the norm,*
- *timeout is a fully-grounded formula that represents the upper-bound waiting condition for the reparation of a violation, taken into account of only after a violation and not before, and*
- $f_n^A, f_n^M, f_n^D, f_n^R, timeout \in \mathscr{L}$.

If, for example, we wanted to model the following norm: "while Ag is driving, he is obliged to not cross in red light, otherwise he will have to pay a fine with cost 100[5] before time is equal to 500", the result would be:

$$n = \langle Ag, \{driving(Ag)\}, \{\neg crossed\text{-}red(Ag, L)\}, \{\neg driving(Ag)\}, \{fine\text{-}paid(100)\}, time(500)\rangle$$

The interpretation of the tuple in Definition 1 is done by means of the deontic formula:

Definition 2. *The deontic interpretation of a norm n, is:*

$$O_{f_n^R \leq timeout}([\alpha \; stit : f_n^M] \preceq f_n^D \mid f_n^A)$$

The syntax of the operator proposed is similar to the obligation operator from other deontic logics, such as dyadic deontic logic and semantics of deadlines, but with important differences. While the $\leq$ used for $f_n^R \leq timeout$ corresponds to the deadline semantics [3] (if *timeout* occurs, there is a permanent violation), the $\preceq$ used in $[\alpha \; stit : f_n^M] \preceq f_n^D$ should rather be read as "$[\alpha \; stit : f_n^M]$ should hold *at all times* at least until f_n^D". Also, the conditional notation $\mid$ used in dyadic deontic logic, which not always has clear semantics in terms of temporality, in the case of the operator proposed $O(A|B)$ should be read as "starting the moment B happens, A should happen" rather than simply "given B, A should happen"[6].

Therefore, the expression shown in Definition 2 is informally read as: *if at some point* f_n^A *holds, agent* α *is obliged to see to it that* f_n^M *is maintained until, at least,* f_n^D holds; *otherwise,* α *is obliged to see to it that* f_n^R *before timeout.* Note that in this informal reading we are not dealing with norm instances yet. How we address this issue, along with the semantics of this obligation operator, will be explained in Subsect. 4.2. Following the example:

$$O_{fine\text{-}paid(100) \leq time(500)}([Ag \; stit : \neg crossed\text{-}red(Ag, L)] \preceq \neg driving(Ag) \mid driving(Ag))$$

informally read as: *if at some point Ag is driving, Ag is obliged to see to it that no red light is crossed until, at least, Ag is not driving anymore; otherwise,*

[5] Each time there is an infraction the fine has to be paid, still, for reasons of simplicity we use a predicate that keeps no track of the different violations

[6] In some works in the literature, this is interpreted as "given B and as long as B happens, A should happen", while in other works it is interpreted in a closer way to our reading

Ag has to pay a fine of 100 before the time is 500. The semantics of this operator are presented in the rest of this section.

4.2 Norm Instances

As previously discussed in Sect. 3, we have to take into account the following issues:

1. deontic statements do not express truth value related to a norm, but rather the existence of a norm [22]; and
2. to check the compliance of a norm, its particular instances must be tracked [27],

Therefore, we need to define the compliance of a norm based on the fulfilment of each of its instantiations. That is, a norm has been complied up to a certain time t if, and only if, each one of the instantiations triggered in times $t_i < t$ have not been violated, where violated means that there has been $\neg f_n^M$ before f_n^D ever happening.

A norm is defined in an abstract manner, affecting all possible participants enacting a given role. In order to work with instances, we need to define a norm instantiation. We consider a substitution θ (we denote it as *substitution instance* when referring to norms) as defined in Sect. 2. Whenever a norm is active, we will say that there is a *norm instance* n^θ for a particular norm n and a substitution instance θ.

Definition 3. *Given a norm n in force and a substitution set θ, we define a norm instance n^θ as $n^\theta = \langle \alpha, \theta(f_n^A), \theta(f_n^M), \theta(f_n^D), timeout \rangle$, where:*

- *$\theta(f_n^A)$, timeout are fully grounded, and*
- *$\theta(f_n^M)$, $\theta(f_n^D)$ may be fully or partially grounded.*

The reason that $\theta(f_n^M)$, $\theta(f_n^D)$ may be partially grounded is that the substitution instance that instantiates the norm – that is, θ such that $\theta(f_n^A)$ holds – is considered in our model to be the sufficient and necessary set of substitutions needed to fully ground f_n^A. It can be the case that the set of variables used in f_n^M and/or f_n^D is larger than the arity of θ. Let us suppose, for example, that the norm should be instantiated at all times while it is *in force*, regardless of any contextual condition: in that case, $f_n^A = \top$. Therefore, we have to assume that a substitution instance θ' for f_n^M or f_n^D should fulfil: $\theta \subseteq \theta'$.

4.3 Norm Lifecycle

Although LTL as a formalism is suitable enough in terms of complexity for reductions to monitoring and planning scenarios, and therefore for practical reasoning from an institutional or individual perspective, there are intrinsic constraints that limit the expressiveness of the framework.

More concretely, the norm instance lifecycle proposed in Fig. 2 cannot be expressed in LTL. As proved in [31], in order to reduce an automata to an LTL

expression – and vice versa –, such automata has to be free of loops that involve more than one state, i.e. only cycles that start and finish in the same state and involve no second state are allowed.

This is an important constraint that prevents our model to have a loop between the (A)ctive and the (V)iolated states. In other words, if we want to use LTL, the lifecycle cannot have cycles that allow to go backwards. Therefore, for the purpose of our formalisation, we propose to adopt the more straightforward lifecycle shown in Fig. 3.

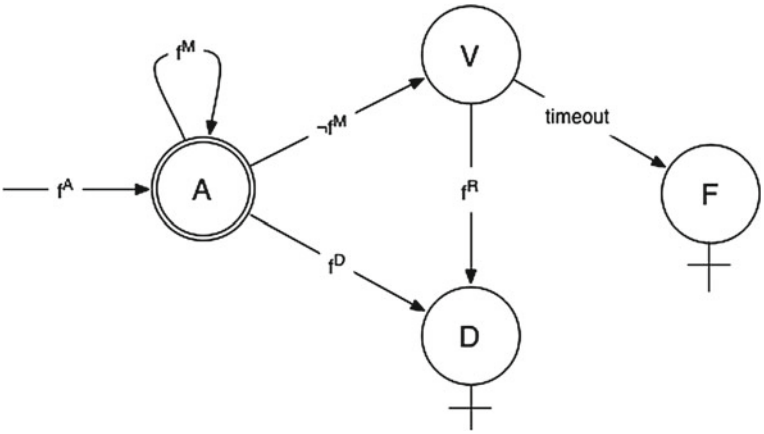

Fig. 3. Self-loop alternating automata-based norm instance lifecycle

The main difference with respect to the automata in Fig. 2 is the handling of violations. As there is no way back to an (A)ctive state anymore, from a (V)iolation state there are only two options: either to repair the norm instance and subsequently (D)eactivate it, or mark it as a (F)ailure if it has not been dealt with for a given amount of time. From an operational perspective, this issue can be worked around by allowing the norm-aware system to create more instances of the same norm if an instance is violated before a deactivation.

For an obligation to have a deontic effect, it is required that the activating condition actually happens at some future point. Additionally, either of the following three conditions should happen:

- The activating condition never occurs so the norm never gets activated.
- Always, between the activating and deactivation condition, the maintenance holds (reached "deactivated" state).
- Maintenance condition holds up to a point where it becomes false and then a violation is permanently raised. In addition, the repair condition occurs later (reached "deactivated" state) before timeout is reached.

In this way we approach most closely that the maintenance of $\theta(f_n^M)$ causes the $\neg viol(n^\theta)$. Thus, the deontic effect of an obligation can be described by the causal effect between the maintenance condition and a violation in Definition 4.

In order to give meaning to the fulfilment of a norm instance, we define a specific operator $\mathscr{O}$ with similar syntax to the abstract norm operator O. Let $M = \{S, \Re, \theta\}$ be an LTL model (using a predicate set $\mathscr{L}$ for the formation of LTL formulas and with θ as described in Subsect. 4.2), $\pi =< s_0, s_1, s_2, \ldots >$ a full path in M, and $viol(n^\theta)$ a predicate belonging to $\mathscr{L}$ representing the violation of a norm instance n^θ, we can establish the semantic relationship between the lifecycle of a norm instance and the fulfilment/violation of a norm as:

Definition 4. Causal semantics for the operator $\mathscr{O}$

$$
\begin{aligned}
M, \pi \models & \mathscr{O}_{\theta(f_n^R) \leq timeout}([\alpha\ stit : \theta(f_n^M)] \preceq \theta(f_n^D) \mid \theta(f_n^A)) \\
& \equiv_{def} \\
M, \pi \models & \mathbf{G}\big(\neg\theta(f_n^A) \wedge \neg viol(n^\theta)\big) \vee \\
& \Big(\mathbf{F}\Big(\theta(f_n^A) \wedge [\forall\theta' : \theta'(\theta(f_n^M))\mathbf{U}\exists\theta'' : \theta''(\theta(f_n^D))]\Big) \wedge \mathbf{G}\neg viol(n^\theta)\Big) \vee \\
& \mathbf{F}\Big(\theta(f_n^A) \wedge [\neg viol(n^\theta)\mathbf{U}\exists\theta' : \neg\theta'(\theta(f_n^M))] \wedge \\
& \quad [\theta'(\theta(f_n^M))\mathbf{U}\big(\neg\theta'(\theta(f_n^M)) \wedge \mathbf{G}viol(n^\theta) \wedge\ (\neg timeout\mathbf{U}\exists\theta'' : \theta''(\theta(f_n^R)))\big)]\Big)
\end{aligned}
$$

The first line of the temporal formula says that the activating condition actually never happens and no violation is raised throughout the executional path. This case does not cause any change in the state of the system. The second line says that there exists some substitution for the activating condition in the future, and that always until a substitution raises an instance of the deactivation condition, the maintenance condition holds for all substitutions. No violation is raised throughout the executional path. This case terminates the norm in a state of *deactivation* (D). The rest of the lines in the formula imply that there exists some substitution for the activating condition in the future, and that at some later point a substitution makes the maintenance condition not hold, thus raising a violation (which remains thereafter). In addition, another substitution makes the repair condition happen at some future after the violation has occurred but before timeout occurs. The norm terminates in a state of *deactivation* (D).

The *failed* state (F), in which the timeout has occurred without the norm having realised the repair condition after a violation, is not described in the formula, since it is an "unwanted" state and should be avoided.

The lifecycle defined in Fig. 3 can be seen as an transition automaton. Transition properties that define how the norm changes its status while events (world changes that modify the predicates' truthness) are occurring can be easily extracted. We are interested in directly representing these transitions as it is useful when dealing with monitoring of norms' status (see Sect. 5.1). The four states *active* (A), *viol* (V), *deactivated* (D), *failed* (F) are described in Definition 5:

Definition 5. Norm lifecycle predicates

$$M, \pi \models \mathbf{X}active(n^\theta) \text{ iff } M, \pi \models (\mathbf{X}\theta(f_n^A) \vee active(n^\theta)) \wedge \mathbf{X}\ \nexists\theta' : \theta'(\theta(f_n^D))$$

$M, \pi \models \mathbf{X}viol(n^\theta)$ *iff* $M, \pi \models active(n^\theta) \wedge \mathbf{X} \nexists \theta' : \theta'(\theta(f_n^M))$

$M, \pi \models \mathbf{X}deactivated(n^\theta)$ *iff*
$(M, \pi \models active(n^\theta) \wedge \mathbf{X}\exists \theta' : \theta'(\theta(f_n^D))) \vee (M, \pi \models viol(n^\theta) \wedge \mathbf{X}\exists \theta' : \theta'(\theta(f_n^R)))$

$M, s \models \mathbf{X}failed(n^\theta)$ *iff* $M, s \models viol(n^\theta) \wedge \mathbf{X}timeout$

The first says that the norm remains in *active* status until there is no instance of deactivation condition occurring. The second says that the norm moves from the *active* to the *viol* state if there is no instance of the maintenance condition. The third says that the norm moves from the *active* to the *deactivated* state if there is an instance of the deactivation condition occurring and that the norm moves from the *viol* to the *deactivated* state if there is an instance of the repair condition occurring. The last says that the norm moves from the *viol* to the *failed* state if timeout occurs.

4.4 From Abstract Norm to Norm Instances

Now we have the apparatus needed to connect the fulfilment of an abstract norm and the fulfilment of its instances, and give semantic meaning to the operator proposed in Definition 2:

Definition 6. Fulfilment of a norm based on the fulfilment of its instances

$M, \pi \models O_{f_n^R \leq timeout}([\alpha\ stit : f_n^M] \preceq f_n^D \mid f_n^A) \equiv_{def}$
$\exists \theta : M, \pi \models \mathbf{F}(\theta(f_n^A)) \Leftrightarrow M, \pi \models \mathscr{O}_{\theta(f_n^R) \leq timeout}([\alpha\ stit : \theta(f_n^M)] \preceq \theta(f_n^D) \mid \theta(f_n^A))$

Informally: *the abstract norm is fulfilled if, and only if, for each possible instantiation of* f_n^A *through time, the obligations of the norm instances activated by* f_n^A *are fulfilled.*

5 Operational Semantics

In this section, we will show how the formalisation proposed in 4 can be reduced to operational semantics that will allow for norm reasoning for two different purposes: the monitoring of normative states from what occurred in the past, from an institutional perspective (see Subsect. 5.1); and the planning of actions in the future taking into account normative constraints, from an agent perspective (see Subsect. 5.2).

5.1 Monitoring Norms

In terms of institutional normative compliance, the detection of normative states is a passive procedure consisting in monitoring past events generated by agents' actions and checking them against a set of active norms. This type of

reasoning can be covered by the declarative aspect of production systems. Using a forward-chaining rule engine, events can automatically trigger normative states – based on a given operational semantics – without requiring a design on how to do it.

Having (1) a direct syntactic translation from norms to rules and (2) a logic implemented in an engine consistent with the process we want to accomplish, allows us to decouple normative state monitoring from the agent reasoning. Our approach is based on creating an initial set of agent- and institutional-independent rules, which the agents – such as manager agents – will be able to transparently query the current normative state at any moment and reason upon it.

In order to achieve this initial set rules, we need to establish a grounding for our formalism. First of all, we will define the lifecycle of a norm instance according to the LTL formalisations of the previous section. We will show how to transform the paths into transition rules, translating the principles of change in normative states into transition rules, effectively reducing our formalisation to a rule-based operational semantics.

In order to track the normative state of an institution at any given point of time, we assume the existence of a knowledge base, in which we will define four sets representing each of the lifecycle states: an active set AS, a violated set VS, a deactivated set DS, and a failed set FS, each of them containing norm instances in the form of tuples: $\{\langle n_i, \theta_j\rangle, \langle n_{i'}, \theta_{j'}\rangle, ..., \langle n_{i''}, \theta_{j''}\rangle\}$.

Definition 7. *A Normative Monitor M_N for a set of norms N is a tuple $M_N = \langle N, AS, VS, DS, FS, S\rangle$, where:*

- *s is the current state of the world, which corresponds to the current path state.*
- *N is the set of norms,*
- $n \in N$, $\langle n, \theta\rangle \in AS \Leftrightarrow M, s \models active(n^\theta)$
- $n \in N$, $\langle n, \theta\rangle \in VS \Leftrightarrow M, s \models viol(n^\theta)$
- $n \in N$, $\langle n, \theta\rangle \in DS \Leftrightarrow M, s \models deactivated(n^\theta)$
- $n \in N$, $\langle n, \theta\rangle \in FS \Leftrightarrow M, s \models failed(n^\theta)$

We denote Γ_{M_N} as the set of all possible configurations of a Normative Monitor M_N.

Definition 8. *The* Transition System TS_{M_N} *for a Normative Monitor M_N is defined by $TS_{M_N} = \langle \Gamma_{M_N}, \rhd\rangle$ where*

- *$\rhd$ is a* transition relation *such that* $\rhd \subseteq \Gamma_{M_N} \times \Gamma_{M_N}$

The inference rules for the transition relation $\rhd$ are described in Fig. 4, where s_i stands for the current state and as, vs, ds, fs correspond to instances of the AS, VS, DS, FS sets of the Normative Monitor tuple.

$$\frac{\theta(f_n^A) \vee \langle n,\theta\rangle \in AS \qquad \neg\theta(f_n^D)}{M_N \rhd \langle N, AS \cup \{\langle n,\theta\rangle\}, VS, DS, FS, s_{i+1}\rangle} \tag{1}$$

$$\frac{\langle n,\theta\rangle \in AS \qquad \neg\theta(f_n^M)}{M_N \rhd \langle N, AS - \{\langle n,\theta\rangle\}, VS \cup \{\langle n,\theta\rangle\}, DS, FS, s_{i+1}\rangle} \tag{2}$$

$$\frac{\langle n,\theta\rangle \in AS \qquad \theta(f_n^D)}{M_N \rhd \langle N, AS - \{\langle n,\theta\rangle\}, VS, DS \cup \{\langle n,\theta\rangle\}, FS, s_{i+1}\rangle} \tag{3}$$

$$\frac{\langle n,\theta\rangle \in VS \qquad \theta(f_n^R)}{M_N \rhd \langle N, AS, VS - \{\langle n,\theta\rangle\}, DS \cup \{\langle n,\theta\rangle\}, FS, s_{i+1}\rangle} \tag{4}$$

$$\frac{\langle n,\theta\rangle \in VS \qquad timeout}{M_N \rhd \langle N, AS, VS - \{\langle n,\theta\rangle\}, DS, FS \cup \{\langle n,\theta\rangle\}, s_{i+1}\rangle} \tag{5}$$

For all cases, $n \in N \wedge n = \langle \alpha, f_n^A, f_n^M, f_n^D, f_n^R, timeout\rangle \wedge \theta \subseteq s_i$ is also part of the transition condition.

Fig. 4. Inference rules for the transition relation $\rhd$: *(1)* Norm instance activated, *(2)* Norm instance violated, *(3)* Norm instance deactivated by fulfilment, *(4)* Norm instance deactivated by reparation, and *(5)* Norm instance failed

By combining these transition rules with the semantics of production systems [32], and additionally transforming the norm condition formulas – normalised in DNF – into rules by means of automatic norm parsing, we obtain a rule-engine which is semantically compliant with our formalism.

Details on the actual implementation have been already presented in [12]. The system designed for this purpose is summarised in Fig. 5 (left). The prototype has been implemented using a combination of XML for norm representation, Java and Clojure for the parsing of the norms, and Drools for the rule engine.

5.2 Planning with Norms

Although the Definition 5 is sufficiently expressive while implementing a monitoring framework, it cannot be applied in a planning system. This is because most planners allow modelling the transitions between states (actions) in a way such that there is exclusive dependency on the values of the previous state's properties. In this way, for example the *active*() status of a norm cannot be easily expressed, since not only does it need to be aware of the activeness at the previous state, but it also needs to be aware of whether at the current state the deactivation condition occurs. The use of extra predicates such as *previous*() and *current* that provide such functionality, allowing to check whether formulas hold in current and previous states, is permitted in some planning frameworks such as TLPlan [17] but it proves to be costly when extensively used.

An alternative, on which we base our implementation is the use of languages that support the use of LTL formulas to restrict the plans produced. Such an attempt is PDDL 3.0 [18]. PDDL 3.0 specification extends PDDL[7] with strong and soft constraints (expressed in LTL formulas) which are imposed on plan

[7] http://planning.cis.strath.ac.uk/competition/

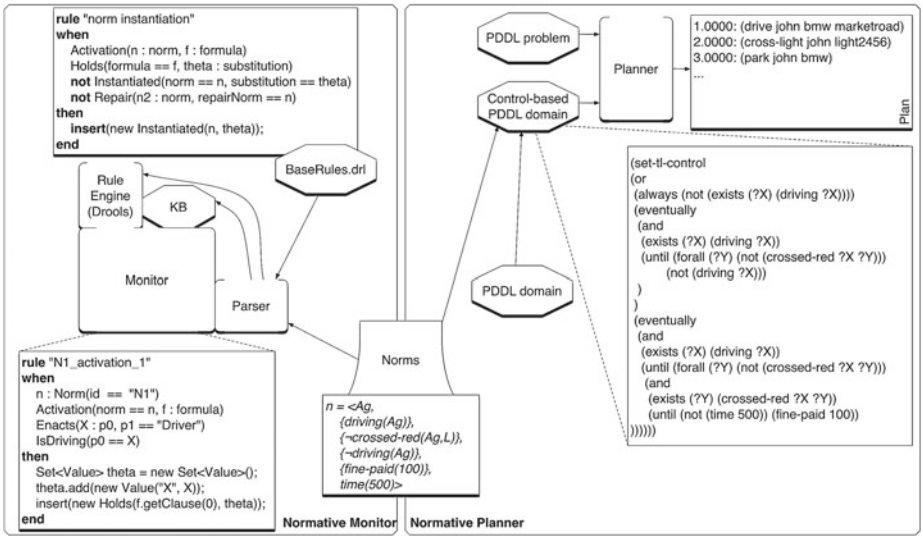

Fig. 5. Normative Monitor and Normative Planner

trajectories, as well as strong and soft problem goals, which are imposed on a plan. TLPlan [17] on the other hand, applies LTL formulas (called *control rules*) to a forward chaining search, reducing in this way the search path by pruning paths that do not comply to the rules. TLPlan is based on (STRIPS-like) semantics that can be easily reduced to PDDL. We choose TLPlan as it seems to contain a complete, robust and rather fast implementation, also allowing extra features such as existential and universal quantifiers. We explain below how the norms are introduced into the planning mechanism.

In order to implement the normative planner, the Definition 4 needs to be transformed into one of use to the planner. While the *viol*() predicate is useful to indicate semantical relation between the norm and its breach, it adds nothing when trying to apply it to a domain, since the violation is actually caused by the progress of the activating, deactivation, and maintenance condition. Therefore, we can eliminate the parts that contain it and create one that we can feed to the planner. The lifecycle then can be represented in Definition 9.

Definition 9. Formula producing norms that reach the deactivated state

$$
\begin{aligned}
M, \pi \models\ & \mathbf{G}(\neg\theta(f_n^A)) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee \\
& \mathbf{F}(\theta(f_n^A) \wedge (\forall\theta' : \theta'(\theta(f_n^M))\mathbf{U}\exists\theta'' : \theta''(\theta(f_n^D)))) \quad \vee \\
& \mathbf{F}(\theta(f_n^A) \wedge (\forall\theta' : \theta'(\theta(f_n^M))\mathbf{U}(\neg\theta'(\theta(f_n^M)) \wedge (\neg timeout\mathbf{U}\exists\theta''' : \theta'''(\theta(f_n^R))))))
\end{aligned}
$$

Thanks to Definition 9 we can represent the norms as control rules within the planning domain. This implies that, for every norm, we need to create such a control rule. The conjunction of all those rules will be the final control rule. We consider that since our norm conditions are defined in DNF form (see Sect. 4), it

is easy to transform them to the appropriate LISP-like format for the planner. For example, a condition $(a(X) \wedge \neg b(X,Y)) \vee c(Z)$ will be transformed into $(or\ (and\ (a\ ?X)\ (b\ ?X\ ?Y))(c\ ?Z))$ Fig. 5 (right) depicts our implementation of the normative planner. The problem file remains intact, while for the set of norms the control rule is created and added to the domain file.

During the execution, the planner will only allow paths where a norm never gets instantiated, or where a norm gets instantiated and never violated or where a norm gets violated but repaired before the specified timeout is reached. Thus, the planner will never allow for a plan that includes a norm instantiation to be violated and never get repaired to be produced. That is, it discards the ones that do not conform to the norm lifecycle. The system allows for multiple instantiations to be checked throughout the executional paths.

TLPlan might take a formula as an input and use it to determine the best plan that optimises it. We can then assign values to the actions that bring about the different norm conditions. Consequently the planner will be able decide and pick between alternative plans that conform to the norm lifecycle (e.g. one that never violates and another that violates and repairs an instance of a norm) while additionally bringing the most profitable outcome for the agent. We will not enter into detail due to lack of space.

We have executed preliminary experiments with TLPlan [17] with up to three norms within a domain. The experiments were run on Mac OSX with Intel Core i7 2.9 Hz processor with 8 GB memory. The results seem promising as in almost all cases, where the outcome was a plan of up to 20 actions and up to 15 instances of norms were created throughout execution, the running time did not significantly increase and remained less than 1 second. This is due to the fact that branches of possible paths get rejected during the forward chaining search. On the other hand, small overhead could be added due to the check of the validity of the control rule on every state, still we have not had any noticeable change on the running time.

6 Conclusions

The resulting semantic framework presented in this paper directly tackles at the same time three important problems related to the practical materialisation of norm-aware systems: clear connection between the deontic level and the operational semantics, the formalisation of explicit norm instances, and the unambiguity of semantic interpretation across implementation domains. We have done so by building, upon diverse previous work, a connection between deontic statements and temporal logics, and between temporal logics to fluents and transition rules. Previous work also shows [12,16] that from the latter representations the translation to the implementation level is also clear. In our case this connection between a single normative specification and two different practical implementations allows us to have a norm monitoring mechanism (used for institutional enforcement) and a norm-aware planning mechanism (used for agent-oriented practical reasoning) that share exactly the same norm semantics (including the

norm lifecycle). This result is vital to ensure that, for instance, the norm enforcement mechanism will state that there is a violation in a case the normative planner found legal and viceversa.

However, this is ongoing research that still needs improvement in several respects and we plan for immediate future work. First of all, we recognise that the constraints on the expressiveness of the norm life-cycle automata from Fig. 3 are quite limiting. We are looking into formalisms that may allow us to work with a version of the life-cycle closer to the one depicted in Fig. 2, probably in a logic framework different from LTL or CTG.

Also, we need to establish the properties of our obligation operator and compare them to the Standard Deontic Logics' properties. Moreover, we are especially interested in defining prohibitions and permissions keeping the syntax of Definition 1.

Section 3 states that there are already formalisations and/or languages that cover some parts of the issues we mention in the Introduction. It is our intention to connect our operational semantics with them.

Additionally, we have been testing our operational semantics with respect to run-time change of norms and normative contexts, and we will extend the norm lifecycle to include new states such as *abrogation* or *derogation*.

Finally, as mentioned in Sect. 3, Searle not only describes regulative rules but also constitutive. We intend to explore the possible implications of adding counts-as rules to our formalisation, following work from [21].

Acknowledgements. This work has been supported by the European funded projects IT-ALIVE (PF-215890) and SUPER HUB (PF-289067). The content of this paper however solely reflect the opinion of the authors, and does not necessarily represent the views of the European Commission.

References

1. Vázquez-Salceda, J.: The role of norms and electronic institutions in multi-agent systems applied to complex domains: the HarmonIA framework. Ph.D. Thesis, 218, January 2003
2. von Wright, G.H.: Deontic logic. Mind, New Series **60**(237), 1–15 (1951)
3. Dignum, F.P.M., Broersen, J., Dignum, V., Meyer, J.-J.: Meeting the deadline: why, when and how. In: Hinchey, M.G., Rash, J.L., Truszkowski, W.F., Rouff, C.A. (eds.) FAABS 2004. LNCS (LNAI), vol. 3228, pp. 30–40. Springer, Heidelberg (2004)
4. Boella, G., van der Torre, L.: Regulative and constitutive norms in normative multiagent systems. In: Proceedings of 10th International Conference on the Principles of Knowledge Representation and Reasoning, KR'04, pp. 255–265 (2004)
5. Aldewereld, H., Grossi, D., Vázquez-Salceda, J., Dignum, F.P.M.: Designing normative behaviour via landmarks. In: Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J.S., Vázquez-Salceda, J. (eds.) ANIREM 2005 and OOOP 2005. LNCS (LNAI), vol. 3913, pp. 157–169. Springer, Heidelberg (2006)

6. García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., Vasconcelos, W.W.: Norm-oriented programming of electronic institutions: a rule-based approach. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS (LNAI), vol. 4386, pp. 177–193. Springer, Heidelberg (2007)
7. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a formalisation of electronic contracting environments. In: Hübner, J.F., Matson, E., Boissier, O., Dignum, V. (eds.) COIN@AAMAS 2008. LNCS, vol. 5428, pp. 156–171. Springer, Heidelberg (2009)
8. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: a model checker for the verification of multi-agent systems. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 682–688. Springer, Heidelberg (2009)
9. Governatori, G., Rotolo, A.: Norm compliance in business process modeling. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 194–209. Springer, Heidelberg (2010)
10. Ågotnes, T., van der Hoek, W., Wooldridge, M.: Robust normative systems and a logic of norm compliance. Logic J. IGPL **18**(1), 4–30 (2010)
11. Criado, N., Argente, E., Noriega, P.: Towards a normative BDI architecture for norm compliance. COIN@ MALLOW2010 (2010)
12. Alvarez-Napagao, S., Aldewereld, H., Vázquez-Salceda, J., Dignum, F.: Normative monitoring: semantics and implementation. In: De Vos, M., Fornara, N., Pitt, J.V., Vouros, G. (eds.) COIN 2010. LNCS, vol. 6541, pp. 321–336. Springer, Heidelberg (2011)
13. López y López, F., Luck, M., d'Inverno, M.: Normative agent reasoning in dynamic societies. In: Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '04, Washington, DC, USA, vol. 2, pp. 732–739 (2004)
14. Kollingbaum, M.J.: Norm-governed practical reasoning agents. Ph.D. Dissertation (2005)
15. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in BDI agents. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '09, vol. 1, pp. 177–184, Richland, SC. (International Foundation for Autonomous Agents and Multiagent Systems) (2009)
16. Panagiotidi, S., Vázquez-Salceda, J.: Norm-aware planning: semantics and implementation. In: 2011 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), pp. 33–36. IEEE, August 2011
17. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artif. Intell. **116**(1–2), 123–191 (2000)
18. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3: the language of the fifth international planning competition. Technical report R.T. 2005–08-07, August 2005
19. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, New York (2004)
20. Searle, J.: The Construction of Social Reality. Free Press, New York (1995)
21. Aldewereld, H., Alvarez-Napagao, S., Dignum, F., Vázquez-Salceda, J.: Making norms concrete. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), pp. 807–814 (2010)
22. Walter, R.: Jörgensen's dilemma and how to face it. Ratio Juris **9**(2), 168–171 (1996)

23. Cardoso, H.L., Oliveira, E.: A context-based institutional normative environment. In: Hübner, J.F., Matson, E., Boissier, O., Dignum, V. (eds.) COIN@AAMAS 2008. LNCS, vol. 5428, pp. 140–155. Springer, Heidelberg (2009)
24. Koo, J.: A Study on the model checking for deontic logic. In: Convergence and Hybrid Information Technology, pp. 832–835 (2008)
25. Prisacariu, C., Schneider, G.: Abstract specification of legal contracts. In: Proceedings of the 12th International Conference on Artificial Intelligence and Law (ICAIL), ACM Request Permissions, pp. 218–219 (2009)
26. Aldewereld, H.: Autonomy vs. conformity: an institutional perspective on norms and protocols. Ph.D. Thesis, Utrecht University (2007)
27. Abrahams, A.S., Bacon, J.M.: The life and times of identified, situated, and conflicting norms. In: Sixth International Workshop on Deontic Logic in Computer, Science (DEON), pp. 3–20 (2002)
28. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2008. LNCS (LNAI), vol. 5397, pp. 1–17. Springer, Heidelberg (2009)
29. Cardoso, H.L., Oliveira, E.: Directed deadline obligations in agent-based business contracts. In: Padget, J., Artikis, A., Vasconcelos, W., Stathis, K., da Silva, V.T., Matson, E., Polleres, A. (eds.) COIN@AAMAS 2009. LNCS, vol. 6069, pp. 225–240. Springer, Heidelberg (2010)
30. Governatori, G.: Representing business contracts in RuleML. Int. J. Coop. Inf. Syst. **14**(2–3), 181–216 (2005)
31. Tauriainen, H.: automata and linear temporal logic: translations with transition-based acceptance. Ph.D. Thesis, Helsinki University of Technology (2006)
32. Cirstea, H., Kirchner, C., Moossen, Michael, M., Moreau, P.E.: Production systems and rete algorithm formalisation. Research report inria-00280938, PROTHEO - INRIA Lorraine - LORIA (2004)