Thomas Ågotnes
Wiebe van der Hoek
Juan A. Rodríguez-Aguilar
Carles Sierra
Michael Wooldridge

A Temporal Logic of Normative Systems

Abstract. We study *Normative Temporal Logic* (NTL), a formalism intended for reasoning about the temporal properties of normative systems. NTL is a generalisation of the well-known branching-time temporal logic CTL, in which the path quantifiers A ("on all paths...") and E ("on some path...") are replaced by the indexed deontic operators O_η ("it is obligatory in the context of the normative system η that ...") and P_η ("it is permissible in the context of the normative system η that..."). After introducing the logic, we give a sound and complete axiomatisation. We then present a symbolic representation language for normative systems, and we identify four different model checking problems, corresponding to whether or not a model is represented symbolically or explicitly, and whether or not we are given a concrete interpretation for the normative systems named in formulae to be model checked. We show that the complexity of model checking varies from P-complete in the simplest case (explicit state model checking where we are given a specific interpretation for all normative systems in the formula) up to EXPTIME-hard in the worst case (symbolic model checking, no interpretation given). We present examples to illustrate the use of NTL, and conclude with discussions of related work (in particular, the relationship of NTL to other deontic logics), and some issues for future work.

Keywords: Normative Systems, Temporal Logic, Multi-Agent Systems.

1. Introduction

Normative systems, or social laws, have been widely promoted as an approach to coordinating multi-agent systems [26, 25, 20, 27, 28, 18]. Crudely, a normative system defines a set of constraints on the behaviour of agents, corresponding to obligations, which may or may not be observed by agents. The designer of a normative system typically has some objective in mind, such that if the constraints of the normative system are observed, then the objective is achieved [18].

A number of formalisms have been proposed for reasoning about normative behaviour in multi-agent systems, typically based on deontic logic [30, 12, 19]. However the computational properties of such formalisms — in particular, their use in the practical design and synthesis of normative systems

David Makinson, Jacek Malinowski and *Heinrich Wansing* (eds.),
Trends in Logic: Towards Mathematical Philosophy

Trends in Logic **27**: 11–48, 2008

and the complexity of reasoning with them — has received relatively little attention. In this paper, we rectify this omission. We present a logic for reasoning about normative systems, which is closely related to the successful and widely-used temporal logic CTL [14]. The idea underpinning *Normative Temporal Logic* (NTL) is to replace the universal and existential path quantifiers of CTL with indexed deontic operators O_η and P_η, where $\mathsf{O}_\eta\varphi$ means that "φ is obligatory in the context of the normative system η", and $\mathsf{P}_\eta\varphi$ means "φ is permissible in the context of the normative system η". Here, φ is a temporal logic expression over the usual CTL temporal operators $\bigcirc, \Diamond, \Box$, and $\mathcal{U}$, and a syntactic construction rule similar to that in CTL applies: every temporal operator must be preceded by a deontic operator. A normative system η is understood to be a set of constraints on the behaviour of agents within the system. In NTL, obligations and permissions are thus, first, contextualised to a normative system η and, second, have a temporal dimension. NTL generalises CTL because by letting $\eta_\emptyset$ denote the empty normative system, which places no constraints on the behaviour of agents, the universal path quantifier A can be interpreted as $\mathsf{O}_{\eta_\emptyset}$. Because of its close relationship to CTL, much of the technical machinery developed for reasoning with CTL can be adapted for use in NTL [14, 11].

The remainder of the paper is structured as follows. After introducing the logic, we give a sound and complete axiomatisation. We then present a symbolic representation language for normative systems. We investigate the complexity of NTL model checking, and identify four different variations of the model checking problem, depending on whether a model is represented symbolically or explicitly, and whether we are given a concrete interpretation for the normative systems named in formulae to be model checked. We show that the complexity of model checking varies from P-complete in the simplest case (explicit state model checking where we are given a specific interpretation for all normative systems in the formula) up to EXPTIME-hard in the worst case (symbolic model checking, no interpretation given). We present two examples to illustrate the use of the logic. We conclude with a discussion of related work, (in particular, a discussion of the relation to other deontic and deontic temporal logics), and some issues for future research.

2. Normative Temporal Logic

2.1. Kripke Structures

Let $\Phi = \{p, q, \ldots\}$ be a finite set of atomic *propositional variables*. A *Kripke structure* (over Φ) is a quadruple

$$\mathcal{K} = \langle S, S^0, R, V \rangle,$$

where:

- S is a finite, non-empty set of *states*, with S^0 being the *initial states* ($\emptyset \subset S^0 \subseteq S$);
- $R \subseteq S \times S$ is a total binary relation on S, which we refer to as the *transition relation*[1]; and
- $V : S \to 2^{\Phi}$ labels each state with the set of propositional variables true in that state.

A *path* over R is an infinite sequence of states $\pi = s_0, s_1, \ldots$ which must satisfy the property that $\forall u \in \mathbb{N}$: $(s_u, s_{u+1}) \in R$. If $u \in \mathbb{N}$, then we denote by $\pi[u]$ the component indexed by u in π (thus $\pi[0]$ denotes the first element, $\pi[1]$ the second, and so on). A path π such that $\pi[0] = s$ is an *s-path*.

2.2. Normative Systems

Normative systems have come to play a major role in multi-agent systems research; for example, under the name of *social laws*, they have been shown to be a useful mechanism for coordination [27]. In this paper, a normative system should be understood simply as *a set of constraints on the behaviour of agents in a system*. More precisely, a normative system defines, for every possible system transition, whether or not that transition is considered to be legal or not, in the context of the normative system. Different normative systems may differ on whether or not a particular transition is considered legal. Formally, a normative system η (w.r.t. a Kripke structure $\mathcal{K} = \langle S, S^0, R, V \rangle$) is simply a subset of R, such that $R \setminus \eta$ is a total relation. We refer to the requirement that $R \setminus \eta$ is total as a *reasonableness* requirement: it prevents social laws which lead to states with no allowed successor. Let $N(R) = \{\eta \mid \eta \subseteq R \ \& \ R \setminus \eta \text{ is total}\}$ be the set of normative systems over R. The intended interpretation of a normative system η is that the presence of an arc (s, s') in η means that the transition (s, s') is forbidden in the context of η, hence, $R \setminus \eta$ denotes the allowed transitions. Since it is assumed that η is reasonable, we are guaranteed that such a transition always exists for every state. If π is a path over R and η is a normative system over R, then we say that π is *η-conformant* if it does not contain any

[1] In the temporal logic literature, it is common to refer to a relation $R \subseteq S \times S$ as being total if $\forall s \in S, \exists s' \in S : (s, s') \in R$.

transition that is forbidden by η, i.e., if $\forall u \in \mathbb{N}$, $(\pi[u], \pi[u+1]) \notin \eta$. We denote the set of η-conformant s-paths (w.r.t. some assumed R) by $\mathcal{C}_\eta(s)$.

Since normative systems in our view are just *sets* (of disallowed transitions), we can *compare* them, to determine, for example, whether one is *more liberal* (less restrictive) than another: if $\eta \subset \eta'$, then η places fewer constraints on a system than η', and hence η is more liberal. Notice that, assuming an *explicit* representation of normative systems, (i.e., representing a normative system η directly as a subset of R), checking such properties can be done in polynomial time. We can also operate on them with the standard set theoretic operations of union, intersection, etc. Taking the union of two normative systems η_1 and η_2 may yield (depending on whether $R \setminus (\eta_1 \cup \eta_2)$ is total) a normative system that is *more restrictive* (less liberal) than either of its parent systems, while taking the *intersection* of two normative systems will yield a normative system which is *less restrictive* (more liberal). The $\cup$ operation is intuitively the act of superposition, or composition of normative systems: imposing one law on top of another. Notice that, when operating on normative systems using such set theoretic operations, care must be taken to ensure the resulting normative system is reasonable.

Example 2.1. *Consider two parallel circular train tracks. At one point both tracks go through the same tunnel. At the east and the west end of the tunnel there are traffic lights, which can be either green or red. A train controller controls the lights. The eastern light should be set to green if and only if there is a train waiting to enter the east end of the tunnel and there is no train waiting at the west end of the tunnel, and similarly for the western light. One train travels on each of the tracks, in opposite directions. We call the train that enters the tunnel at the eastern end the east train and the other train the west train. Obviously, the trains should not enter the tunnel if the light is red.*

We can model this situation by considering the physical properties and the normative properties separately*, as Kripke structures and normative systems respectively. We assume that each train can be in one of three states:* tunnel *(the train is in the tunnel);* waiting *(the train is waiting to enter the tunnel);* away *(the train is neither in the tunnel nor waiting). When away, the train can either be away or waiting in the next state; when waiting the train can either be waiting or in the tunnel in the next state; when the train is in the tunnel it leaves the tunnel and is away in the next state. Thus, we use propositional atoms* $eTunnel, eWaiting, eAway, wTunnel, wWaiting, wAway$ *to encode the position of the east and west train. We also use atoms eGreen and wGreen to represent the fact that the eastern/western lights are green.*

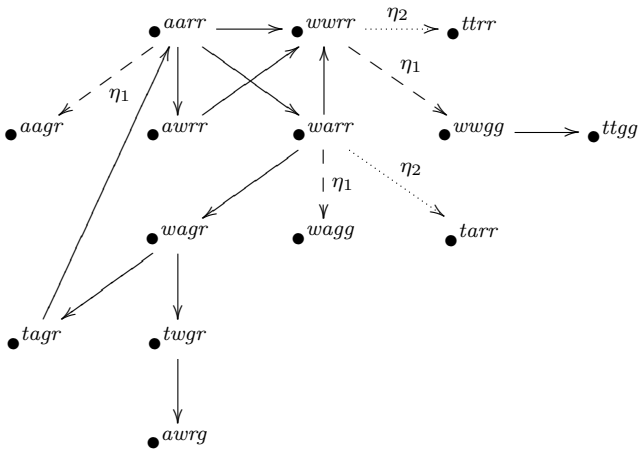

Figure 1. Kripke model of the trains example, including all physically possible transitions. Only a part of the model is shown. The transitions prohibited by the normative systems η_1 and η_2 are shown with dashed and dotted lines, respectively. The labelling of the states is abbreviated for readability: "twgr" stands for tunnel-waiting-green-red and means that *wTunnel*, *eWaiting*, *wGreen* are true and that all other atoms (including *eGreen*) are false.

Thus, $\neg$eGreen means that the eastern light is red, and so on. Let $\mathcal{K}$ be the Kripke structure where the states correspond to all possible configurations of the atomic propositions, the (single) initial state is the state where both lights are red and both trains away, and the transitions are all physically possible transitions — illustrated in Figure 1. The transitions include entering on a red light, but exclude physically impossible transitions such as a train going directly from the tunnel *state to the* waiting *state.*

Let η_1 be the normative system corresponding to the normative requirement on the switching of the lights described above: η_1 contains all transitions between states s_1 and s_2 in which one of the lights are set to green (in s_2) without *the appropriate condition (as explained above) being true in s_1. The normative system η_1 is illustrated by labels on the transitions in Figure 1. The description above contains another normative requirement as well: trains should only enter the tunnel on a green light. Let η_2 be the normative system corresponding to that requirement: η_2 contains all transitions between states s_1 and s_2 such that a train is in the tunnel in s_2 only if the corresponding light is green in s_1. It is easy to see that $\eta_1, \eta_2 \in N(R)$, where R is the transition relation of $\mathcal{K}$.*

Finally, while the norms in this particular example are designed to avoid a crash, there are other problems, such as "deadlock" (both trains can wait forever for a green light), which they do not avoid. For simplicity, we will only consider the norms mentioned above.

2.3. Syntax of NTL

The language of NTL is a generalisation of CTL: the only issue that may cause confusion is that, within this language, we refer explicitly to normative systems, which are of course *semantic* objects. We will therefore assume a stock of syntactic elements Σ_η which will denote normative systems. An *interpretation* for symbols Σ_η with respect to a transition relation R is a function $I : \Sigma_\eta \to N(R)$. When R is a transition relation of Kripke structure $\mathcal{K}$ we say that I is an interpretation over $\mathcal{K}$. We will assume that the symbol $\eta_\emptyset$ always denotes the "emptyset" normative system, i.e., the normative system which forbids *no* transitions. Note that this normative system will be well-defined for *any* Kripke structure. Thus, we require that all interpretations I satisfy the property that $I(\eta_\emptyset) = \emptyset$. If the interpretation function I is clear from context or not relevant, we will sometimes identify the symbol η with the normative system it denotes.

The syntax of NTL is defined by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{P}_\eta \bigcirc \varphi \mid \mathsf{P}_\eta(\varphi \,\mathcal{U}\, \varphi) \mid \mathsf{O}_\eta \bigcirc \varphi \mid \mathsf{O}_\eta(\varphi \,\mathcal{U}\, \varphi)$$

where $p \in \Phi$ is a propositional variable and $\eta \in \Sigma_\eta$ denotes a normative system. Sometimes we call α occurring in an expression $\mathsf{O}_\eta\alpha$ or $\mathsf{P}_\eta\alpha$ a *temporal formula* (although such an α is not a well-formed formula of NTL).

2.4. Semantic Rules

The semantics of NTL are given with respect to the satisfaction relation "$\models$". $\mathcal{K}, s \models_I \varphi$ holds when $\mathcal{K}$ is a Kripke structure, s is a state in $\mathcal{K}$, I an interpretation over $\mathcal{K}$, and φ a formula of the language, as follows:

$\mathcal{K}, s \models_I \top$;

$\mathcal{K}, s \models_I p$ iff $p \in V(s)$ (where $p \in \Phi$);

$\mathcal{K}, s \models_I \neg\varphi$ iff not $\mathcal{K}, s \models_I \varphi$;

$\mathcal{K}, s \models_I \varphi \vee \psi$ iff $\mathcal{K}, s \models_I \varphi$ or $\mathcal{K}, s \models_I \psi$;

$\mathcal{K}, s \models_I \mathsf{O}_\eta \bigcirc \varphi$ iff $\forall \pi \in \mathcal{C}_{I(\eta)}(s) : \mathcal{K}, \pi[1] \models_I \varphi$;

$\mathcal{K}, s \models_I \mathsf{P}_\eta \bigcirc \varphi$ iff $\exists \pi \in \mathcal{C}_{I(\eta)}(s) : \mathcal{K}, \pi[1] \models_I \varphi$;

$\mathcal{K}, s \models_I \mathsf{O}_\eta(\varphi \,\mathcal{U}\, \psi)$ iff $\forall \pi \in \mathcal{C}_{I(\eta)}(s), \exists u \in \mathbb{N}$, s.t. $\mathcal{K}, \pi[u] \models_I \psi$ and $\forall v, (0 \leq v < u) : \mathcal{K}, \pi[v] \models_I \varphi$

$\mathcal{K}, s \models_I \mathsf{P}_\eta(\varphi \,\mathcal{U}\, \psi)$ iff $\exists \pi \in \mathcal{C}_{I(\eta)}(s), \exists u \in \mathbb{N}$, s.t. $\mathcal{K}, \pi[u] \models_I \psi$ and $\forall v, (0 \leq v < u) : \mathcal{K}, \pi[v] \models_I \varphi$

The remaining classical logic connectives ("$\wedge$", "$\rightarrow$", "$\leftrightarrow$") are assumed to be defined as abbreviations in terms of $\neg$ and $\vee$, in the conventional manner. We define the remaining CTL-style operators for $\Diamond$ and $\Box$ as abbreviations:

$$\begin{array}{rcl} \mathsf{O}_\eta \Diamond \varphi & \equiv & \mathsf{O}_\eta(\top \mathcal{U} \varphi) \\ \mathsf{P}_\eta \Diamond \varphi & \equiv & \mathsf{P}_\eta(\top \mathcal{U} \varphi) \\ \mathsf{O}_\eta \Box \varphi & \equiv & \neg \mathsf{P}_\eta \Diamond \neg \varphi \\ \mathsf{P}_\eta \Box \varphi & \equiv & \neg \mathsf{O}_\eta \Diamond \neg \varphi \end{array}$$

Recalling that $\eta_\emptyset$ denotes the empty normative system, we obtain the conventional path quantifiers of CTL [14] as follows:

$$\begin{array}{rcl} \mathsf{A}\alpha & \equiv & \mathsf{O}_{\eta_\emptyset}\alpha \\ \mathsf{E}\alpha & \equiv & \mathsf{P}_{\eta_\emptyset}\alpha \end{array}$$

Thus the CTL universal path quantifier can be understood as obligation in the context of the empty normative system, which places no restictions on which transitions the system takes, while the existential path quantifier can be understood as permission in the context of this normative system.

We write $\mathcal{K} \models_I \varphi$ if $\mathcal{K}, s_0 \models_I \varphi$ for all $s_0 \in S^0$, $\mathcal{K} \models \varphi$ if $\mathcal{K} \models_I \varphi$ for all I, and $\models \varphi$ if $\mathcal{K} \models \varphi$ for all $\mathcal{K}$.

EXAMPLE 2.2 (Example 2.1 continued). *Let $\mathcal{K}, \eta_1, \eta_2$ be as in example 2.1. Let I be such that $I(\eta_1) = \eta_1$, $I(\eta_2) = \eta_2$, $I(\eta_3) = \eta_1 \cup \eta_2$ (it is easy to see that also $\eta_1 \cup \eta_2 \in N(R)$). Let the formula*

$$crash = eTunnel \wedge wTunnel$$

denote a crash situation. We have that (recall that $\mathcal{K} \models_I \varphi$ means that φ is satisfied in all the initial states of $\mathcal{K}$ under I):

- $\mathcal{K} \models_I \mathsf{O}_{\eta_1} \bigcirc \neg wGreen$. *In the initial state, according to normative system η_1 it is obligatory that the western light stays red in the next state.*
- $\mathcal{K} \models_I \mathsf{P}_{\eta_1}(\neg eGreen \,\mathcal{U}\, eTunnel)$. *$\eta_1$ permits the eastern light to stay red until the east train is in the tunnel.*
- $\mathcal{K} \models_I \neg \mathsf{P}_{\eta_2}(\neg eGreen \,\mathcal{U}\, eTunnel)$. *$\eta_2$ does not permit the eastern light to stay red until the east train is in the tunnel.*
- $\mathcal{K} \models_I \mathsf{O}_{\eta_1} \Box(wGreen \rightarrow \neg eGreen)$. *It is obligatory in the context of η_1 that at least one of the lights are red.*
- $\mathcal{K} \models_I \mathsf{P}_{\eta_\emptyset} \Diamond crash$. *Without any constraining norms, the system permits a crash in the future.*

- $\mathcal{K} \models_I \mathsf{P}_{\eta_1} \Diamond \mathit{crash}$. *The normative system* η_1 *permits a crash.*
- $\mathcal{K} \models_I \mathsf{O}_{\eta_3} \Box \neg \mathit{crash}$. *It is obligatory, in the context of normative system* η_3*, that a crash never occurs;* η_3 *does not permit a crash at any point in the future.*

The following are examples of expressions involving nested operators. It is worth reflecting on the compositional meaning of nested operators. For example, $\mathsf{P}_{\eta_3} \Diamond \mathsf{P}_{\eta_1} \bigcirc \mathit{crash}$ *means that* η_3 *permits a computation along which in some future state* $\mathsf{P}_{\eta_1} \bigcirc \mathit{crash}$ *is true. However, in the evalutation of* $\mathsf{P}_{\eta_1} \bigcirc \mathit{crash}$ *in states along that computation, the system is* not *restricted by* η_3 *(but only by* η_1*).*

- $\mathcal{K} \models_I \mathsf{O}_{\eta_\emptyset} \Box((\mathit{wWaiting} \wedge \neg \mathit{wGreen}) \rightarrow \neg \mathsf{P}_{\eta_2} \bigcirc \mathit{wTunnel})$. *It is obligatory in the system that it is always the case that if the west train is waiting and the western light is red then the western train is not permitted by* η_2 *in the tunnel in the next state.*
- $\mathcal{K} \models_I \mathsf{P}_{\eta_2} \Diamond \mathsf{P}_{\eta_3} \bigcirc \mathit{crash}$. η_2 *permits a future state where a crash in the next state is permitted even by* η_3*.*
- $\mathcal{K} \models_I \mathsf{P}_{\eta_3} \Diamond \mathsf{P}_{\eta_1} \bigcirc \mathit{crash}$. η_3 *permits a future state where a crash in the next state is permitted by* η_1*.*
- $\mathcal{K} \models_I \mathsf{O}_{\eta_3} \Box \mathsf{O}_{\eta_2} \bigcirc \neg \mathit{crash}$. η_3 *does not permit a future state where a crash is permitted in the next state by* η_2*.*

2.5. Properties and Axiomatisation

The following Proposition makes precise the expected property that *a less liberal system has more obligations and less permissions than a more liberal system.*

PROPOSITION 2.3. *Let* $\mathcal{K}$ *be a Kripke structure,* I *an interpretation over* $\mathcal{K}$ *and* $\eta_1, \eta_2 \in \Sigma_\eta$*.*

$$\textit{If } I(\eta_1) \subseteq I(\eta_2) \textit{ then } \mathcal{K} \models_I \mathsf{O}_{\eta_1}\varphi \rightarrow \mathsf{O}_{\eta_2}\varphi \textit{ and } \mathcal{K} \models_I \mathsf{P}_{\eta_2}\varphi \rightarrow \mathsf{P}_{\eta_1}\varphi$$

We now go on to exhaustively describe the universally valid properties, of NTL as well as some derived systems, by presenting sound and complete axiomatisations.

First, let NTL$^-$ be NTL without the empty normative system. Formally, NTL$^-$ is defined exactly as NTL, except for the requirement that Σ_η contains the $\eta_\emptyset$ symbol and the corresponding restriction on interpretations. An

(Ax1) All validities of propositional logic

(Ax2) $\mathsf{P}_\eta \Diamond \varphi \leftrightarrow \mathsf{P}_\eta(\top \mathcal{U} \varphi)$

(Ax2b) $\mathsf{O}_\eta \Box \varphi \leftrightarrow \neg \mathsf{P}_\eta \Diamond \neg \varphi$

(Ax3) $\mathsf{O}_\eta \Diamond \varphi \leftrightarrow \mathsf{O}_\eta(\top \mathcal{U} \varphi)$

(Ax3b) $\mathsf{P}_\eta \Box \varphi \leftrightarrow \neg \mathsf{O}_\eta \Diamond \neg \varphi$

(Ax4) $\mathsf{P}_\eta \bigcirc (\varphi \vee \psi) \leftrightarrow (\mathsf{P}_\eta \bigcirc \varphi \vee \mathsf{P}_\eta \bigcirc \psi)$

(Ax5) $\mathsf{O}_\eta \bigcirc \varphi \leftrightarrow \neg \mathsf{P}_\eta \bigcirc \neg \varphi$

(Ax6) $\mathsf{P}_\eta(\varphi \mathcal{U} \psi) \leftrightarrow (\psi \vee (\varphi \wedge \mathsf{P}_\eta \bigcirc \mathsf{P}_\eta(\varphi \mathcal{U} \psi)))$

(Ax7) $\mathsf{O}_\eta(\varphi \mathcal{U} \psi) \leftrightarrow (\psi \vee (\varphi \wedge \mathsf{O}_\eta \bigcirc \mathsf{O}_\eta(\varphi \mathcal{U} \psi)))$

(Ax8) $\mathsf{P}_\eta \bigcirc \top \wedge \mathsf{O}_\eta \bigcirc \top$

(Ax9) $\mathsf{O}_\eta \Box(\varphi \rightarrow (\neg\psi \wedge \mathsf{P}_\eta \bigcirc \varphi)) \rightarrow (\varphi \rightarrow \neg \mathsf{O}_\eta(\gamma \mathcal{U} \psi))$

(Ax9b) $\mathsf{O}_\eta \Box(\varphi \rightarrow (\neg\psi \wedge \mathsf{P}_\eta \bigcirc \varphi)) \rightarrow (\varphi \rightarrow \neg \mathsf{O}_\eta \Diamond \psi)$

(Ax10) $\mathsf{O}_\eta \Box(\varphi \rightarrow (\neg\psi \wedge (\gamma \rightarrow \mathsf{O}_\eta \bigcirc \varphi))) \rightarrow (\varphi \rightarrow \neg \mathsf{P}_\eta(\gamma \mathcal{U} \psi))$

(Ax10b) $\mathsf{O}_\eta \Box(\varphi \rightarrow (\neg\psi \wedge \mathsf{O}_\eta \bigcirc \varphi)) \rightarrow (\varphi \rightarrow \neg \mathsf{P}_\eta \Diamond \psi$

(Ax11) $\mathsf{O}_\eta \Box(\varphi \rightarrow \psi) \rightarrow (\mathsf{P}_\eta \bigcirc \varphi \rightarrow \mathsf{P}_\eta \bigcirc \psi)$

(R1) If $\vdash \varphi$ then $\vdash \mathsf{O}_\eta \Box \varphi$ (generalization)

(R2) If $\vdash \varphi$ and $\vdash \varphi \rightarrow \psi$ then $\vdash \psi$ (modus ponens)

(Obl) $\mathsf{O}_{\eta_\emptyset} \alpha \rightarrow \mathsf{O}_\eta \alpha$

(Perm) $\mathsf{P}_\eta \alpha \rightarrow \mathsf{P}_{\eta_\emptyset} \alpha$

Figure 2. The two systems NTL$^-$ ((Ax1)–(R2), derived from an axiomatisation of CTL [14]) and NTL ((Ax1)–(R2),(Obl),(Perm)). α stands for a temporal formula.

axiom system for NTL$^-$, denoted $\vdash^-$, is defined by axioms and rules (Ax1)–(R2) in Figure 2. NTL$^-$ can be seen as a *multi-dimensional* variant of CTL, where there are several indexed versions of each path quantifier[2]. Indeed, the axiomatisation has been obtained from an axiomatisation of CTL [14].

Going on to NTL, we add axioms (Obl) and (Perm) (Figure 2); the corresponding inference system is denoted $\vdash$. We then (by soundness, see below), have the following chain of implications in NTL (the second element in the

[2]Semantically, we can view the NTL structures as multi-dimensional CTL structures with one (total) transition relation $R \setminus I(\eta)$ for each normative system. This definition of multi-dimensional structures is different from *multiprocess temporal structures* as defined in [5, 14]. In the latter, only the *union* of the transition relations is required to be total.

chain is a variant of a deontic axiom discussed later). If something is naturally, or physically inevitable, then it is obligatory in any normative system; if something is an obligation within a given normative system η, then it is permissible in η; and if something is permissible in a given normative system, then it is naturally (physically) possible:

$$\models (\mathsf{A}\varphi \rightarrow \mathsf{O}_\eta\varphi) \qquad \models (\mathsf{O}_\eta\varphi \rightarrow \mathsf{P}_\eta\varphi) \qquad \models (\mathsf{P}_\eta\varphi \rightarrow \mathsf{E}\varphi)$$

THEOREM 2.4 (Soundness and Completeness).
For every φ in the language of NTL$^-$*, we have $\models \varphi$ iff $\vdash^- \varphi$. The same holds for $\vdash$ with respect to formulas from* NTL.

PROOF. (Sketch.) Soundness is straightforward.

For completeness, consider first NTL$^-$. Let φ_0 be a consistent formula. As noted earlier, we can view NTL$^-$ as a multi-dimensional extension of CTL. Rather than extending the tableau-based method for proving the completeness of CTL in [14], we describe[3] a construction which employs the CTL completeness result directly, viewing a formula as a CTL formula for one dimension δ at a time by reading O_δ and P_δ as CTL path quantifiers A and E, respectively, and treating formulae starting with a δ'-operator ($\delta' \neq \delta$) as atomic formulae. By completeness of CTL, we get a CTL model for the formula (if it is consistent), where the states are labelled with atoms such as $\mathsf{O}_{\delta'}\Gamma$ or $\mathsf{P}_{\delta'}\Gamma$ (for $\delta' \neq \delta$). Then, for each δ' and each state, we expand the state by taking the conjunction of η'-formulae the state is labelled with, construct a (single-dimension) CTL model of that formula, and "glue" the root of the model together with the state. Repeat for all dimensions and all states.

In order to keep the formulae each state is labelled with finite, we consider only subformulae of φ_0. A δ-atom is a subformula of φ_0 starting with either P_δ or O_δ. Let $At^{-\delta}$ denote the union all of δ'-atoms for all $\delta' \neq \delta$. Furthermore, we assume that φ_0 is such that every occurrence of $\mathsf{P}_\eta(\alpha_1 \,\mathcal{U}\, \alpha_2)$ ($\mathsf{O}_\eta(\alpha_1 \,\mathcal{U}\, \alpha_2)$) is immediately preceeded by $\mathsf{P}_\eta\bigcirc$ ($\mathsf{O}_\eta\bigcirc$) — we call this *XU form*. Any formula can be rewritten to XU form by recursive use of the axioms (Ax6) and (Ax7). We start with a model with a single state labelled with the literals in a consistent disjunct of φ_0 written on disjunctive normal form. We continue by expanding states labelled with formulae, one dimension δ at a time. In general, let $at(\delta, s)$ be the union of the set of δ-atoms s is labelled with and the set of negated δ-atoms of XU form s is *not* labelled

[3]Due to lack of space we cannot give all the technical details here. For the interested reader, more details are available at `http://home.hib.no/ansatte/tag/misc/mctl.pdf`.

with. We can now view $\bigwedge at(\delta, s)$ as a CTL formula over a language with primitive propositions $\Phi \cup At^{-\delta}$. $\bigwedge at(\delta, s)$ is NTL$^-$ consistent. The following holds: any NTL$^-$ consistent formula is satisfied by a state s' in a CTL model M' viewing $\Phi \cup At^{-\delta}$ as primitive propositions, such that for any $\delta' \neq \delta$ and any state t of M', $\bigwedge(\delta', t)$ is NTL$^-$ consistent, and s' does not have any ingoing transitions (the proof is left for the reader). This ensures that we can "glue" the pointed model M', s' to the state s while labelling the transitions in the model with the dimension δ we expanded – M', s' satisfies the formulae needed to be true there. The fact that s' does not have any ingoing transitions ensures that we can append M', s' to s without changing the truth of δ-atoms at s'. The fact that φ_0 is of XU form ensures that all labelled formulae are of XU form, which again ensures that we don't add new labels to a state when we expand it (because all the formulae we expand start with a next-modality). The fact that $\bigwedge(\delta', t)$ is consistent for states t in the expanded model, ensures that we can repeat the process. Only a finite number of repetitions are needed, depending on the number of nested operators of different dimensions in the formula, after which we can remove the non-Φ labels without affecting the truth of φ_0 and obtain a proper model.

The same construction is used for NTL, treating $\eta_\emptyset$ as any other dimension, with the following difference. When expanding a node along dimension δ, when gluing the CTL model to the expanded node label the transitions with $\eta_\emptyset$ in addition to δ. Axioms (Obl) and (Perm) ensure that this is consistent with the $\eta_\emptyset$-atoms present at the node. ■

Going beyond NTL, we can impose further structure on Σ_η and its interpretations. For example, we can extend the logical language with basic statements like $\eta \equiv \eta'$ and $\eta \sqsubset \eta'$ ($\sqsubseteq$ can then be defined), with the obvious interpretation. Furthermore, we can add unions and intersections of normative systems by requiring Σ_η to include symbols $\eta \sqcup \eta'$, $\eta \sqcap \eta'$ whenever it includes η and η', and require interpretations to interpret $\sqcup$ as set union and $\sqcap$ as set intersection. As discussed earlier, we must then further restrict interpretations such that $R \setminus (I(\eta_1) \cup I(\eta_2))$ is always total. This would give us a kind of calculus of normative systems. Let $\mathcal{K}$ be a Kripke structure and I be an interpretation with the mentioned properties:

$$\mathcal{K} \models_I \mathsf{P}_{\eta \sqcup \eta'} \varphi \to \mathsf{P}_\eta \varphi \qquad \mathcal{K} \models_I \mathsf{P}_\eta \varphi \to \mathsf{P}_{\eta \sqcap \eta'} \varphi$$
$$\mathcal{K} \models_I \mathsf{O}_\eta \varphi \to \mathsf{O}_{\eta \sqcup \eta'} \varphi \qquad \mathcal{K} \models_I \mathsf{O}_{\eta \sqcap \eta} \varphi \to \mathsf{O}_\eta \varphi$$

(these follow from Proposition 2.3). Having such a calculus allows one to reason about the composition of normative systems, similar to the way one constructs complex programs from simpler ones in Dynamic Logic [16].

Of course we could drop the reasonableness constraint. This would make it possible that "too many" norms (i.e., too many constraints on agent behaviour) may prevent *any* transition from a given state.

3. Symbolic Representations

Our aim is for NTL to be used in the formal specification and analysis of normative systems. To this end, we envisage a computer program that will take as input a Kripke structure $\mathcal{K}$, representing some system of interest, together with an NTL formula φ representing a query about this system, and some normative systems I; the program will then determine whether or not the property expressed by φ holds of $\mathcal{K}, I$, i.e., whether or not $\mathcal{K} \models_I \varphi$. Such a program is called a *model checker* [11]. However, this raises the issue of exactly how the Kripke structure $\mathcal{K}$ and normative systems I are presented to the model checker. One possibility is to simply list all the states, the propositions true in these states, and the transitions in the transition relation. Such a representation is called an *explicit state* representation. In practice, explicit state representations of Kripke structures are almost never used. This is because of the *state explosion problem*: given a system with n Boolean variables, the system will typically have 2^n states, and so an explicit representation in the input is not practicable. Instead, practical reasoning tools provide *succinct*, *symbolic* representation languages for defining Kripke structures. In this section, we present such a language for defining models, and also introduce an associated symbolic language for defining normative systems[4].

3.1. A Symbolic Language for Models

The REACTIVE MODULES LANGUAGE (RML) was introduced by Alur and Henzinger as a simple but expressive formalism for specifying game-like distributed system models [2], and this language is used as the model specification language for several model checkers [4]. In this section, we consider a "stripped down" version of RML called SIMPLE REACTIVE MODULES LANGUAGE (SRML), introduced in [17]; this language represents the core of RML,

[4]Notice that when we refer to a "symbolic representation", we are referring to the use of a symbolic definition of the Kripke structure *in the input to the model checker*; however, the term "symbolic model checking" is also commonly used to refer to the internal representation used by a model checker, and in this paper, we are not concerned with this issue [11].

with some "syntactic sugar" removed to keep the presentation (and semantics) simple.

Here is an example of an agent in SRML (note that agents are referred to as "modules" in SRML):

$$
\begin{array}{l}
\texttt{module } \textit{toggle} \texttt{ controls } x \\
\quad \texttt{init} \\
\quad \ell_1 : \top \rightsquigarrow x' := \top \\
\quad \ell_2 : \top \rightsquigarrow x' := \bot \\
\quad \texttt{update} \\
\quad \ell_3 : x \rightsquigarrow x' := \bot \\
\quad \ell_4 : (\neg x) \rightsquigarrow x' := \top
\end{array}
$$

This module, named *toggle*, controls a single Boolean variable, x. Occurrences of the primed version x' refer to the fresh initial value of x (in `init`) or its value in the next state (`update`). The *choices* available to the agent at any given time are defined by those `init` and `update` rules[5]. The `init` rules define the choices available to the agent with respect to the initialisation of its variables, while the `update` rules define the agent's choices subsequently. In this example, there are two `init` rules and two `update` rules. The `init` rules define two choices for the initialisation of this variable: assign it the value $\top$ (i.e., "true") or the value $\bot$ (i.e., "false"). Both of these rules can fire initially, as their conditions ($\top$) are always satisfied; in fact, only one of the available rules will ever *actually* fire, corresponding to the "choice made" by the agent on that decision round. On the left hand side of the rules are *labels* (ℓ_i) which are used to identify the rules. Note that labels do not form part of the original RML language, and in fact play no part in the semantics of RML — their role will become clear below. We assume a distinguished label "`[]`"; the role of this label will also become clear below. With respect to `update` rules, the first rule says that if x has the value $\top$, then the corresponding choice is to assign it the value $\bot$, while the second rules says that if x has the value $\bot$, then it can subsequently be assigned the value $\top$. In other words, the module non-deterministically chooses a value for x initially, and then on subsequent rounds toggles this value. Notice that in this example, the `init` rules of this module are non-deterministic, while the `update` rules are deterministic: SRML (and RML) allow for non-determinism in both initialisation and update rules. An SRML *system* is a set of such modules.

[5]To be more precise, the rules are in fact *guarded commands*.

Formally, a rule γ over a set of propositional variables Φ and a set of labels $\mathcal{L}$ is an expression

$$\ell : \varphi \rightsquigarrow v_1' := \psi_1; \ldots; v_k' := \psi_k$$

where $\ell \in \mathcal{L}$ is a label, φ (the guard) is a propositional logic formula over Φ, each v_i is a member of Φ and ψ_i is a propositional logic formula over Φ. We require that no variable v_i appears on the l.h.s. of two assignment statements in the same rule (hence no issue on the ordering of the updates arises). The intended interpretation is that if the formula φ evaluates to true against the interpretation corresponding to the current state of the system, then the rule is *enabled* for execution; executing the statement means evaluating each ψ_i against the current state of the system, and setting the corresponding variable v_i to the truth value obtained from evaluating ψ_i. We say that $v_1, \ldots, v_k$ are the *controlled variables* of γ, and denote this set by $ctr(\gamma)$. A set of rules is said to be *disjoint* if their controlled variables are mutually disjoint.

When dealing with the SRML representation of models, a state is simply equated with a propositional valuation (i.e., the set of states of an SRML system is exactly the set of possible valuations to variables within it: $S = 2^{\Phi}$)[6]. Given a state $s \subseteq \Phi$ and a rule $\gamma : \varphi \rightsquigarrow v_1' := \psi_1; \ldots; v_k' := \psi_k$ such that s enables γ (i.e., $s \models \varphi$) we denote the result of *executing* γ on s by $s \oplus \gamma$. For example, if $s = \{p, r\}$, and $\gamma = p \rightsquigarrow q' := \top; r' := p \wedge \neg r$, then $s \oplus \gamma = \{p, q\}$. Note that if a variable does not have its value defined explicitly by a rule that is enabled in some state, then this variable is assumed to remain unchanged.

Given a state $s \subseteq \Phi$, and set Γ of disjoint rules over Φ such that every member of Γ is enabled by s, then the interpretation s' resulting from Γ on s is denoted by $s' = s \oplus \Gamma$ (since the members of Γ are disjoint, we can pick them in any order to execute on s).

As described above, there are two classes of rules that may be declared in a module: `init` and `update`. An `init` rule is only used to initialise the

[6]Thus the state space of an SRML system will be exponential in the number of variables in the system. One may then wonder how this squares with our requirement earlier that we want to avoid an representation for models that is overly large. The point is that while we cannot ultimately escape the fact that the number of possible states in a system will be exponential in the number of variables, if we want to *reason* about a system, then we still need some compact way of representing the system. This is the exactly the role played by S(RML). It provides a compact language for *defining* Kripke structures, and is suitable for use as the input to a model checker. A language which was not compact in this way would be useless in practice as the input language to a model checker, since the size of the input would be unfeasibly large.

values of variables, when the system begins execution. We will assume that the guards to `init` rule are "$\top$", i.e., every `init` rule is enabled for execution in the initialisation round of the system.

An SRML *module*, m, is a triple:

$$m = \langle ctr, init, update \rangle \text{ where:}$$

- $ctr \subseteq \Phi$ is the (finite) set of variables controlled by m;
- $init$ is a (finite) set of *initialisation* rules, such that for all $\gamma \in init$, we have $ctr(\gamma) \subseteq ctr$; and
- $update$ is a (finite) set of *update* rules, such that for all $\gamma \in update$, we have $ctr(\gamma) \subseteq ctr$.

Note that this definition permits variables to be unitialised by the `init` rules of the module. Such variables are by default assumed to be initialised to $\bot$.

Given a module m, we denote the controlled variables of m by $ctr(m)$, the initialisation rules of m by $init(m)$, and the update rules of m by $update(m)$. An SRML system ρ is then an $(n+2)$-tuple

$$\rho = \langle Ag, \Phi, m_1, \ldots, m_n \rangle$$

where $Ag = \{1, \ldots, n\}$ is a set of agents, Φ is a vocabulary of propositional variables, and for each $i \in Ag$, m_i is the corresponding module defining i's choices; we require that $\{ctr(m_1), \ldots, ctr(m_n)\}$ forms a partition of Φ (i.e., every variable in Φ is controlled by some agent, and no variable is controlled by more than one agent).

A *joint* rule (j.r.) is an indexed tuple $\langle \gamma_1, \ldots, \gamma_n \rangle$ of rules, with a rule $\gamma_i \in m_i$ for each $i \in Ag$. A j.r. $\langle \gamma_1, \ldots, \gamma_n \rangle$ is enabled by a propositional valuation s iff all its members are enabled by s.

It is straightforward to extract the Kripke structure $\mathcal{K}_\rho = \langle S_\rho, S^0_\rho, R_\rho, V_\rho \rangle$ corresponding to an SRML system ρ:

- the initial states S^0_ρ correspond to the valuations that could be generated by the `init` rules of ρ against the empty valuation;
- the remaining states in S_ρ are those that could be generated by some sequence of enabled update joint rules from some initial state;
- the transition relation R_ρ is defined by $(s, s') \in R_\rho$ iff there exists some update j.r. $\langle \gamma_1, \ldots, \gamma_n \rangle$ such that this j.r. is enabled in s and $s' = s \oplus \{\gamma_1, \ldots, \gamma_n\}$.

Notice that there is nothing in this definition which requires a Kripke structure $\mathcal{K}_\rho$ corresponding to a normative system ρ to be reasonable: it is the responsibility of the modeller, defining a normative system using SRML, to ensure this.

3.2. A Symbolic Language for Normative Systems

We now introduce the SRML *Norm Language* (SNL) for representing normative systems, which corresponds to the SRML language for models. The general form of a normative system definition in SNL is as follows:

$$\begin{array}{l}\texttt{normative-system} \quad id\\ \quad \chi_1 \texttt{ disables } \ell_{1_1}, \ldots, \ell_{1_k}\\ \quad \ldots\\ \quad \chi_m \texttt{ disables } \ell_{m_1}, \ldots, \ell_{m_k}\end{array}$$

Here, $id \in \Sigma_\eta$ is the name of the normative system; these names will be used to refer to normative systems in formulae of NTL. The body of the normative system is defined by a set of *constraint rules*. A constraint rule

$$\chi \texttt{ disables } \ell_1, \ldots, \ell_k$$

consists of a condition part χ, which is a propositional logic formula over the propositional variables Φ of the system, and a set of rule labels $\{\ell_1, \ldots, \ell_k\} \subseteq \mathcal{L}$. The intuition is that if χ is satisfied in a particular state, then any SRML rule with a label that appears on the r.h.s. of the constraint rule will be disabled in that state, according to this normative system. Consider the following simple example.

$$\begin{array}{l}\texttt{normative-system} \quad forceTrue\\ \quad \top \texttt{ disables } \ell_3\end{array}$$

We here define a normative system *forceTrue*, which consists of a single rule. The condition part of the rule is $\top$, and hence always fires; in this case, the effect is to disable the rule with label ℓ_3. Since the condition part of this rule is always enabled, in the *forceTrue* normative system, rule ℓ_3 can never fire.

EXAMPLE 3.1 (Example 2.1 continued). *The following* SRML *modules describe the Kripke model $\mathcal{K}$ from Example 2.1.*

module wtrain controls $wAway, wWaiting, wTunnel$
init
$[] : \top \rightsquigarrow wAway' := \top; wWaiting' := \bot; wTunnel := \bot$
update
$Wwait : wAway \vee wWaiting \rightsquigarrow wWaiting' := \top; wAway' := \bot$
$Wstayaway : wAway \rightsquigarrow wAway' := \top$
$Wenter : wWaiting \rightsquigarrow wWaiting' := \bot; wTunnel' := \top$
$Wleave : wTunnel \rightsquigarrow wTunnel' := \bot; wAway := \top$

The module for the western train controls the variables describing its position. The four update rules correspond to the physical actions available. The module etrain for the eastern train is defined in the same way, with rules named Eenter and so on.

$$
\begin{array}{l}
\texttt{module}\ \mathit{controller}\ \texttt{controls}\ \mathit{wGreen}, \mathit{eGreen} \\
\quad \texttt{init} \\
\quad [] : \top \ \leadsto\ \mathit{wGreen}' := \bot; \mathit{eGreen} := \bot \\
\quad \texttt{update} \\
\quad \mathit{RR} : \top \ \leadsto\ \mathit{wGreen}' := \bot; \mathit{eGreen}' := \bot \\
\quad \mathit{RG} : \top \ \leadsto\ \mathit{wGreen}' := \bot; \mathit{eGreen}' := \top \\
\quad \mathit{GR} : \top \ \leadsto\ \mathit{wGreen}' := \top; \mathit{eGreen}' := \bot \\
\quad \mathit{GG} : \top \ \leadsto\ \mathit{wGreen}' := \top; \mathit{eGreen}' := \top
\end{array}
$$

The controller module controls the variables describing the lights. For every update the module chooses one of the rules corresponding to the four possible light settings.

The following SNL *specifications describe the normative systems η_1 and η_2.*

$$
\begin{array}{l}
\texttt{normative-system} \quad \eta_1 \\
\quad (\neg \mathit{wWaiting} \vee \mathit{eWaiting})\ \texttt{disables}\ \mathit{GR}, \mathit{GG} \\
\quad (\neg \mathit{eWaiting} \vee \mathit{wWaiting})\ \texttt{disables}\ \mathit{RG}, \mathit{GG}
\end{array}
$$

$$
\begin{array}{l}
\texttt{normative-system} \quad \eta_2 \\
\quad \neg \mathit{wGreen}\ \texttt{disables}\ \mathit{Wenter} \\
\quad \neg \mathit{eGreen}\ \texttt{disables}\ \mathit{Eenter}
\end{array}
$$

Formal Definition of SNL

Formally, an SNL constraint rule is a pair

$$c = \langle \varphi, L \rangle$$

where φ is a propositional formula over Φ, and $L \subseteq \mathcal{L}$ is a set of rule labels. An SNL normative system is then a pair

$$\eta = \langle id, C \rangle$$

where $id \in \Sigma_\eta$ is a unique identifier for the normative system and C is a set of SRML constraint rules. In any given state s, the set of SRML rules that are disabled by a normative system $\langle id, C \rangle$ will be the set of rules whose labels appear on the right hand side of constraint rules in C whose condition part is satisfied in s.

Given SNL normative systems η_1 and η_2, for some SRML system ρ, we say: η_1 is *at least as liberal* as η_2 in system ρ if for every state $s \in S_\rho$, every rule that is enabled according to η_2 is enabled according to η_1; and they are *equivalent* if for every state $s \in S_\rho$, the set of rules enabled according to η_1 and η_2 are the same.

THEOREM 3.2. *The problem of testing whether* SNL *normative system* η_1 *is at least as liberal as* SNL *normative system* η_2 *is* PSPACE*-complete, as is the problem of testing equivalence of such systems.*

PROOF. We do the proof for checking equivalence; the liberality case is similar. For membership of PSPACE, consider the complement problem: guess a state s, check that $s \in S_\rho$, (reachability of states in RML is in PSPACE [2]) and check that there is some rule enabled in s according to η_2 is not enabled in s according to η_1, or vice versa. Hence the complement problem is in NPSPACE, and so the problem is in PSPACE. For PSPACE-hardness, we reduce the problem of propositional invariant checking over (S)RML modules [2]. Given an SRML system ρ and propositional formula φ, define normative systems η_1 and η_2 as follows (where ℓ does not occur in ρ):

η_1	η_2
$\neg\varphi$ `disables` ℓ	$\bot$ `disables` ℓ

According to η_2, ℓ is always enabled; thus η_1 will be equivalent to η_2 iff φ holds across all reachable states of the system. ■

4. Model Checking

The model checking problem is an important computational problem for any logic, since model checking is perhaps the most successul approach to the automated verification of logical properties of systems [11]. We consider two versions of the model checking problem for NTL, depending on whether the model is presented explicitly or symbolically. For each of these cases, there are two further possibilities, depending on whether we are given an interpretation I for normative systems named in formulae or not. If we are given an interpretation for the normative systems named in the formula, then NTL model checking essentially amounts to a conventional model checking problem: showing that, under the given interpretation, the model and associated normative systems have certain properties. However, the *uninterpreted* model checking problem corresponds to the *synthesis* of normative systems: we ask whether *there exist* normative systems that would have the desired properties. Thus the uninterpreted model checking problems combine model checking with a satisfiability checking aspect.

4.1. Explicit State Model Checking

The *interpreted explicit state model checking problem* for NTL is as follows.

> Given a Kripke structure $\mathcal{K} = \langle S, S^0, R, V \rangle$, interpretation $I : \Sigma_\eta \rightarrow N(R)$ and formula φ of NTL, is it the case that $\mathcal{K} \models_I \varphi$?

It is known that the model checking problem for CTL may be solved in time $O(|\mathcal{K}| \cdot |\varphi|)$ [14], and is in fact P-complete [22]. The standard dynamic programming algorithm for CTL model checking may be trivially adapted for interpreted explicit state NTL model checking, and may be seen to have the same time complexity. More interesting perhaps is the case where we are *not* given an interpretation. The *uninterpreted explicit state model checking problem* for NTL is as follows.

> Given a Kripke structure $\mathcal{K} = \langle S, S^0, R, V \rangle$ and formula φ of NTL, does there exist an interpretation $I : \Sigma_\eta \rightarrow N(R)$ such that $\mathcal{K} \models_I \varphi$?

Notice that uninterpreted model checking has a very natural application, as follows. We have a Kripke structure $\mathcal{K}$ and want a normative system η that will ensure some property, so we write an NTL formula φ, which refers to η, describing this property (the property might, for example, be $\mathsf{O}_\eta \Box \neg \mathit{fail}$); the uninterpreted model checking problem then corresponds to the *feasibility* problem described in [18]: it asks whether there in fact exist a normative system that has the desired properties. We can show:

THEOREM 4.1. *The uninterpreted explicit state model checking problem for* NTL *is* NP*-complete.*

PROOF. For membership in NP, simply guess an interpretation I and verify that $\mathcal{K} \models_I \varphi$. Since interpretations are polynomial in the size of the Kripke structure and formula, guessing can be done in (nondeterministic) polynomial time, and checking is the interpreted explicit state model checking problem. Hence the problem is in NP. For NP-hardness, we reduce SAT. Given a SAT instance $\varphi(x_1, \ldots, x_k)$, we create an instance of the uninterpreted explicit state model checking problem as follows. For each propositional variable x_i in the SAT instance, we create two variables $t(x_i)$ and $f(x_i)$, and we define a Kripke structure with $3k + 1$ states, as illustrated in Figure 3; state s_0 is the initial state, and state s_{3k+1} is a final state, with the only transition possible from this state being back to itself. Now, given the input SAT instance $\varphi(x_1, \ldots, x_k)$, we denote by $\varphi^*(x_1, \ldots, x_k)$ the NTL formula obtained by systematically replacing every propositional variable x_i with $\mathsf{P}_\eta \Diamond t(x_i)$. Finally, we define the formula to be model checked as the

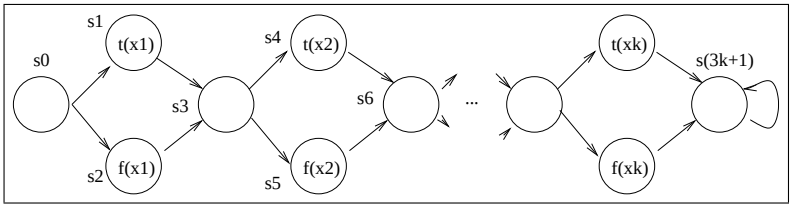

Figure 3. Reduction for Theorem 4.1.

conjunction of the following formulae.

$$\varphi^*(x_1, \ldots, x_k) \tag{I}$$

$$\bigwedge_{1 \leq i \leq k} (\mathsf{P}_\eta \Diamond t(x_i) \rightarrow \neg \mathsf{P}_\eta \Diamond f(x_i)) \tag{II}$$

$$\bigwedge_{1 \leq i \leq k} (\mathsf{P}_\eta \Diamond f(x_i) \rightarrow \neg \mathsf{P}_\eta \Diamond t(x_i)) \tag{III}$$

$$\bigwedge_{1 \leq i \leq k} (\mathsf{P}_\eta \Diamond (t(x_i) \vee f(x_i))) \tag{IV}$$

If this formula is satisfied in the structure by some interpretation, then the interpretation for η must give a satisfying valuation for $\varphi(x_1, \ldots, x_k)$; conversely, if $\varphi(x_1, \ldots, x_k)$ is satisfiable, then any satisfying assignment defines an interpretation for η that makes the formula true in the structure. ■

4.2. Symbolic Model Checking

As we noted above, explicit state model checking problems are perhaps of limited interest, since such representations are exponentially large in the number of propositional variables. Thus we now consider the SRML *model checking problem for* NTL. Again, we have two versions, depending on whether we are given an interpretation or not. The interpreted version is as follows:

> Given an SRML system ρ, a set of SNL normative systems $I = \{\eta_1, \ldots, \eta_k\}$ acting as an interpretation, and an NTL formula φ in which the only normative systems named are defined in I, is it the case that $\mathcal{K}_\rho \models_I \varphi$?

THEOREM 4.2. *The interpreted* SRML *model checking problem for* NTL *is* PSPACE*-complete.*

PROOF. PSPACE-hardness is by a reduction from the problem of propositional invariant verification for SRML, which is proved PSPACE-complete

in [1][7]. Given a propositional formula φ and an (S)RML system ρ, let $I = \{\eta_\emptyset\}$, and simply check whether $\rho \models_I \mathsf{O}_{\eta_\emptyset} \Box\varphi$. Membership of PSPACE is by adapting the CTL symbolic model checking algorithm of Cheng [10]. ■

The *uninterpreted* SRML model checking problem for NTL is defined exactly as expected:

> Given an SRML system ρ and an NTL formula φ, does there exist a set of SNL normative systems $I = \{\eta_1, \ldots, \ldots, \eta_k\}$, one for each η named in φ, such that $\mathcal{K}_\rho \models_I \varphi$?

This problem is provably worse (under standard complexity theoretic assumptions) than the interpreted version.

THEOREM 4.3. *The uninterpreted* SRML *model checking problem for* NTL *is* EXPTIME*-hard.*

PROOF. We prove EXPTIME-hardness by reduction from the problem of determining whether a given player has a winning strategy in the two-player game PEEK-G_4 [29, p.158]. An instance of PEEK-G_4 is a quad:

$$\langle X_1, X_2, X_3, \varphi \rangle$$

where:

- X_1 and X_2 are disjoint, finite sets of Boolean variables, with the intended interpretation that the variables in X_1 are under the control of agent 1, and X_2 are under the control of agent 2;
- $X_3 \subseteq (X_1 \cup X_2)$ are the variables deemed to be true in the initial state of the game; and
- φ is a propositional logic formula over the variables $X_1 \cup X_2$, representing the winning condition.

The game is played in a series of rounds, with the agents $i \in \{1, 2\}$ alternating (with agent 1 moving first) to select a value (true or false) for one of their variables in X_i, with the game starting from the initial assignment of truth values defined by X_3. Variables that were not changed retain the same truth value in the subsequent round. An agent wins in a given round if it makes a move such that the resulting truth assignment defined by that round makes the winning formula φ true. The decision problem associated with PEEK-G_4 involves determining whether agent 2 has a winning strategy in a given game instance $\langle X_1, X_2, X_3, \varphi \rangle$. Notice that PEEK-$G_4$ only requires "memoryless" (Markovian) strategies: whether or not an agent i can win

[7] In fact, the result of [2] is for RML in general, but the proof does not rely on any features of RML that are not present in SRML.

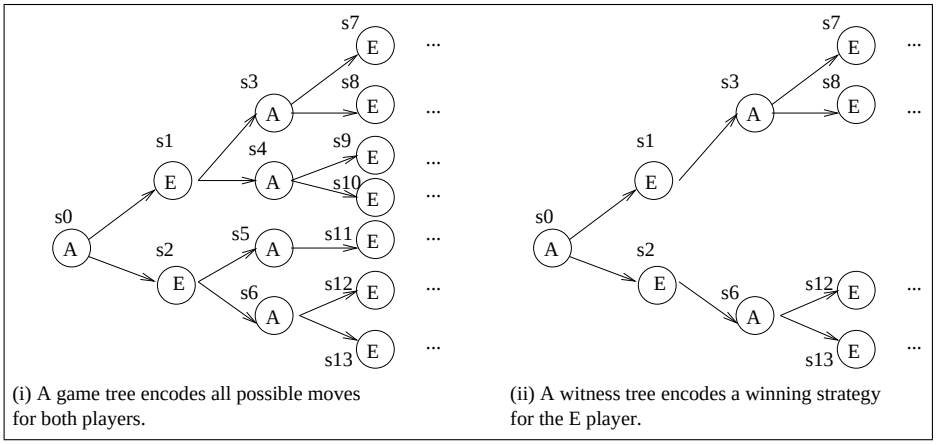

Figure 4. Game trees and witness trees.

depends only on the current truth assignment, the distribution of variables, the winning formula, and whose turn it is currently. As a corollary, if agent i can force a win, then it can force a win in $O(2^{|X_1 \cup X_2|})$ moves.

The idea of the proof is as follows. We can understand the possible plays of a finite two player game of perfect information as a tree (see Figure 4(i)), where nodes correspond to configurations of the game, and are choice points for the two players A (universal) and E (existential). Thus in an A node the universal player makes a choice, while in an E node the existential player makes a choice. We are interested in whether the E player has a winning strategy in such a game. If this is the case, then there will be a witness to this in the form of a sub-tree of the game tree, which characterises all plays of a winning strategy for E. This witness tree will be a sub-tree of the game tree with the following characteristics (see Figure 4):

- The starting position of the game is present in the witness tree.
- At every A node, all outgoing arcs of the game tree from this node must be present in the witness tree. (The E player strategy must win against all possible A moves.)
- At every E node, there can be only one outgoing node. (The E player's strategy can select only one move in any given state.)
- Every play in the witness tree must correspond to a win for the E player — that is, every possible infinite path through the witness tree from the starting position must contain a node in which the E player wins.

The idea of the reduction is to define an SRML system such that the computations of this system correspond to the plays of the PEEK-G_4 instance,

and then define a NTL formula referring to a single normative system η, such that η will encode a witness tree for player 2.

Formally, given an instance $\langle X_1, X_2, X_3, \varphi \rangle$ of PEEK-G_4, we produce an instance of SRML model checking as follows. For each Boolean variable $x \in (X_1 \cup X_2)$, we create a variable with the same name in our SRML model, and we create an additional Boolean variable *turn*, with the intended interpretation that if $turn = \top$, then it is agent 1's turn to move, while if $turn = \bot$, then it is agent 2's turn to move. We have a module *move*, the purpose of which is to control *turn*, toggling its value in each successive round, starting from the initial case of it being agent 1's move.

```
module move controls turn
  init
  ▯⊤ ⇝ turn′ := ⊤
  update
  ▯turn ⇝ turn′ := ⊥
  ▯(¬turn) ⇝ turn′ := ⊤
```

For each of the two PEEK-G_4 players $i \in \{1, 2\}$, we create an SRML module ag_i that `controls` the variables X_i. The module ag_i is as follows. It begins by deterministically initialising the values of all its variables to the values defined by X_3 (that is, if variable $x \in X_i$ appears in X_3 then this variable is initialised to $\top$, otherwise it is initialised to $\bot$). Subsequently, when it is this player's turn, it can non-deterministically choose at most one of the variables under its control and toggle the value of this variable; when it is not this player's turn, it has no choice but to do nothing, leaving the value of all its variables unchanged. The general structure of ag_1 is thus as follows, where $X_1 = \{x_1, \ldots, x_k\}$.

```
module ag1 controls x1, ..., xk
  init
  // initialise to values from X3
  ▯⊤ ⇝ x1′ := ...; xk := ...
  update
  ℓ1_1 : turn ⇝ x1′ := ⊥
  ℓ1_2 : turn ⇝ x1′ := ⊤
  ...
  ℓ1_2k : turn ⇝ xk′ := ⊥
  ℓ1_2k+1 : turn ⇝ xk′ := ⊤
  ℓ1_2k+2 : ⊤ ⇝ skip
```

Notice that an agent can always `skip`, electing to leave its variables unchanged; and, if it is not this agent's turn to move, this is the *only* choice it has. Agent ag_2 has a similar structure.

We now define the formula to model check. First, we define $chng(x_i)$ to mean that variable i changes value in some transition according to η:

$$chng(x_i) \equiv ((x_i \wedge \mathsf{P}_\eta \bigcirc \neg x_i) \vee (\neg x_i \wedge \mathsf{P}_\eta \bigcirc x_i))$$

Agent 2 is an existential player: if it is agent 2's turn, then at most one of its possible moves is allowed in the witness tree[8].

$$\mathsf{O}_\eta \Box (\sum_{x_i \in X_2} chng(x_i) \leq 1)$$

If the E player changes the value of one of its variables, then this change is implemented in all its next states.

$$\bigwedge_{x_i \in X_2} \begin{array}{l} \mathsf{O}_\eta \Box(\neg turn) \rightarrow \\ (chng(x_i) \rightarrow ((\mathsf{P}_\eta \bigcirc x_i \leftrightarrow \mathsf{O}_\eta \bigcirc x_i) \wedge (\mathsf{P}_\eta \bigcirc \neg x_i \leftrightarrow \mathsf{O}_\eta \bigcirc \neg x_i)))] \end{array}$$

If the E player leaves the value of one of its variables unchanged in one next state, then it is unchanged in all its next states.

$$\bigwedge_{x_i \in X_2} \begin{array}{l} \mathsf{O}_\eta \Box(\neg turn) \rightarrow \\ (\neg chng(x_i) \rightarrow ((x_i \leftrightarrow \mathsf{O}_\eta \bigcirc x_i) \wedge (\neg x_i \leftrightarrow \mathsf{O}_\eta \bigcirc \neg x_i)))] \end{array}$$

Agent 1 is a universal player: all of its possible moves must be in the witness tree.

$$\mathsf{O}_\eta \Box turn \rightarrow \left[\bigwedge_{x_i \in X_1} chng(x_i) \right]$$

Finally, the runs that remain must represent wins for agent 2:

$$\mathsf{O}_\eta(\neg\varphi) \mathcal{U} (\varphi \wedge turn)$$

Conjoining these formulae gives the formula to model check. We claim that this formula passes iff agent 2 has a winning strategy. For suppose that the formula passes. Then η defines a witness tree for agent 2. That it corresponds to a well-defined strategy follows from the other properties: for example, agent 2 is only alllowed to make one choice in any given state when it is it's turn, which must win against all choices of agent 1. Similarly, if agent 2 has a winning strategy, then this strategy corresponds to a normative system η such that the formula passes under this interpretation. ■

[8] We use the $\sum$ notation here as an abbreviation for the obvious propositional equivalent.

5. Case Study: Traffic Control

We use a simple case study to illustrate some of the concepts we have introduced. The basis of the case study is as follows:

> Consider a circular road with two parallel lanes. Vehicles circulate on the two lanes clockwise. We consider three types of vehicles: cars, taxis, and ambulances. Each of the lanes is discretised into m positions, each position possibly occupied by a vehicle. In what follows, lane 1 stands for the outer lane, while lane 0 stands for the inner lane. We will refer to lane 0 as the right lane and to lane 1 as the left lane considering the direction of the vehicles. At each time step, cars and taxis can either stand still or change their position by one unit ahead, possibly changing lane at the same time. For instance a car could go from position 5 on the left lane to either position 6 on the right lane or position 6 on the left lane — or it could choose to stand still. Ambulances can stand still or change their position by one or two units, either straight or changing lanes at will.

To avoid crashes and make it possible for ambulances to get to hospitals faster, and to give taxis priority over private cars, we can imagine a number of norms that regulate the behaviour of the vehicles:

η_1 : Ambulances have priority over all other vehicles. By this we mean, in more detail, that other vehicles should stop whenever there is an ambulance behind them.

η_2 : Cars cannot use the rightmost (priority) lane.

η_3 : Vehicles have "right" priority. By this we mean that a vehicle should stop if there is another vehicle to its right. This is, of course, a very strict rule for prioritarisation which we adopt for simplicity. Otherwise, in order for a car to give way to another car with right priority, a signalling system should be used.

η_4 : Ambulances give "priority" to ambulances ahead. By an ambulance giving priority to ambulances ahead, we mean that the ambulance slows down (and therefore can only change its position by at most one unit) when there is another ambulance one unit right in front of it. Thus, norm η_4 is intended to avoid ambulances crashing when they are close and take 2-unit moves.

These norms act on the decisions that agents can make by constraining them. For instance, η_1 will force cars to stop in order to allow ambulances to overtake them.

Now, our goal in the remainder of this section is to show how the technical tools developed in this article can be used to analyse this scenario. A full scale "implementation" (involving *real* taxis and ambulances, etc) is beyond our current resources. However, what we can do instead is to take the key features of the scenario, as described above, and model them in SRML and SNL, abstracting away from lower level implementation details. We can then use model checking tools to investigate the properties of the scenario. Note that we cannot be certain that the results we obtain truly reflect reality; however, the models of systems and normative systems that we develop can, we believe, usefully inform subsequent development, and can help to identify potential issues with normative systems at an early stage of design.

Vehicle Modules

We model each vehicle as a module containing the rules that determine their physically legal movements. We define two types of modules, one for each of the two types of vehicles: those with 2-unit speed and those with 1-unit speed. Cars and taxis are vehicles of 1-unit speed, while ambulances are vehicles of 2-unit speed.

Assume that there are v vehicles named $1, \ldots, v$, each of which is either a car, a taxi or an ambulance. It is assumed that there are more positions than vehicles ($m > v$). It might be the case that none of the vehicles are cars, and the same for taxis and ambulances. Let $cars = \{c_0, \ldots, c_q\} \subseteq \{1, \ldots, v\}$ ($q \geq 0$) be the (names of) the cars. Similarly, $taxis = \{t_0, \ldots, t_r\}$ and $ambu = \{a_0, \ldots, a_s\}$. We first describe the Boolean variables. For each vehicle $1 \leq i \leq v$, position $1 \leq pos \leq m$ and $lane \in \{0, 1\}$, we have a Boolean variable $vpos_i(lane, pos)$. For example, $vpos_2(1, 7)$ means that vehicle number 2 is in position 7 on the left lane.

For each car $i \in cars$ we define a module *car-i* as in Figure 5 (for simplicity, the set $\{v'_1 := \psi_1, \ldots, v'_k := \psi_k\}$ is used as an abbreviation for the sequence of assignment operations $v'_1 := \psi_1; \ldots; v'_k := \psi_k$.)

Each car module controls the variables describing its own position. The initial position of car i is at position i along one of the tracks; the track is chosen non-deterministically (in what follows the inital positions of the cars are completely arbitrary, this particular choice was made as a simple way to ensure that two cars don't occupy the same position). In the update phase, a car module can perform one of four actions. First, the car can stand still, in which case there are no changes to the controlled variables (in this case the right hand side of the $still_i$ rule is just a dummy expression indicating no change). Second, the car can move straight ahead. In all of the three rules

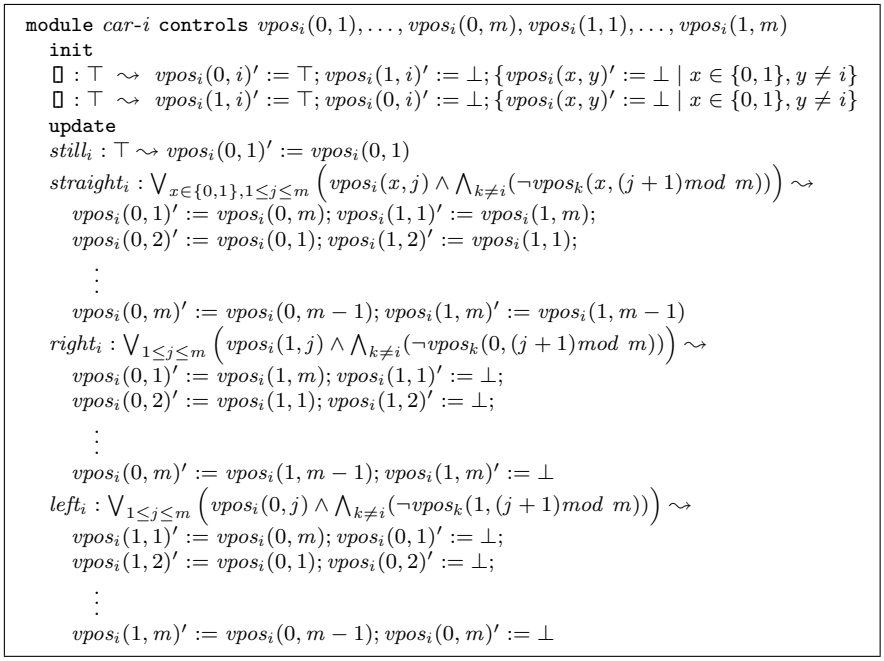

Figure 5. Cars in the ambulance scenario.

which move the car it is assumed that a car will only move to a position which is currently not occupied. This is a reasonable safety assumption about behaviour of cars, however it is neither sufficient (two cars might move simultaneously to the same position) nor necessary (the car currently occupying the position might move to another position at the same time) to avoid crashes. Alternatively, this assumption could have been implemented as a separate normative system. Thus, the guard of the $straight_i$ rule ensures that the position immediately in front of car i is currently available. The right hand side of this rule updates the position of car i by setting $vpos_i(x, y+1)$ to true if $vpos_i(x, y)$ were true before the rule was executed, and so on. The guard of the $right_i$ rule checks that the car is in the left lane and that one position ahead in the right lane is available, and the r.h.s. updates the position. Similarly for $left_i$. Note that the operations on vehicles' positions are modulo-m operations, where m is the number of positions in the road.

We similarly define a module *taxi-i* for each taxi $i \in taxis$ — see Figure 6.

Ambulances follow the same schema except that they have two-step rules which (also) can only be executed when the road is clear. We define a module *ambu-i* for each ambulance $i \in ambu$ — see Figure 7.

module *taxi-i* **controls** $vpos_i(0,1),\ldots,vpos_i(0,m),vpos_i(1,1),\ldots,vpos_i(1,m)$
  init
    (as for car-i)
  update
    (as for car-i)

Figure 6. Taxis in the ambulance scenario.

module *ambu-i* **controls** $vpos_i(0,1),\ldots,vpos_i(0,m),vpos_i(1,1),\ldots,vpos_i(1,m)$
  init
    (as for car-i)
  update
  $still_i$: *(as for car-i)*
  $straight_i$: *(as for car-i)*
  $right_i$: *(as for car-i)*
  $left_i$: *(as for car-i)*
  $straightstraight_i : \bigvee_{x\in\{0,1\},1\leq j\leq m}$
    $\Big(vpos_i(x,j)\wedge\bigwedge_{k\neq i}(\neg vpos_k(x,(j+1)mod\ m)\wedge\neg vpos_k(x,(j+2)mod\ m))\Big)\rightsquigarrow$
    $vpos_i(0,1)'\coloneqq vpos_i(0,m-1); vpos_i(1,1)'\coloneqq vpos_i(1,m-1);$
    $vpos_i(0,2)'\coloneqq vpos_i(0,m); vpos_i(1,2)'\coloneqq vpos_i(1,m);$
      $\vdots$
    $vpos_i(0,m)'\coloneqq vpos_i(0,m-2); vpos_i(1,m)'\coloneqq vpos_i(1,m-2)$
  $straightright_i : \bigvee_{1\leq j\leq m}$
    $\Big(vpos_i(1,j)\wedge\bigwedge_{k\neq i}(\neg vpos_k(1,(j+1)mod\ m)\wedge\neg vpos_k(0,(j+2)mod\ m))\Big)\rightsquigarrow$
    $vpos_i(0,1)'\coloneqq vpos_i(1,m-1); vpos_i(1,1)'\coloneqq \bot;$
    $vpos_i(0,2)'\coloneqq vpos_i(1,m); vpos_i(1,2)'\coloneqq \bot;$
      $\vdots$
    $vpos_i(0,m)'\coloneqq vpos_i(1,m-2); vpos_i(1,m)'\coloneqq \bot$
  $rightstraight_i : \bigvee_{1\leq j\leq m}$
    $\Big(vpos_i(1,j)\wedge\bigwedge_{k\neq i}(\neg vpos_k(0,(j+1)mod\ m)\wedge\neg vpos_k(0,(j+2)mod\ m))\Big)\rightsquigarrow$
    $vpos_i(0,1)'\coloneqq vpos_i(1,m-1); vpos_i(1,1)'\coloneqq \bot;$
    $vpos_i(0,2)'\coloneqq vpos_i(1,m); vpos_i(1,2)'\coloneqq \bot;$
      $\vdots$
    $vpos_i(0,m)'\coloneqq vpos_i(1,m-2); vpos_i(1,m)'\coloneqq \bot$
  $straightleft_i : \bigvee_{1\leq j\leq m}$
    $\Big(vpos_i(0,j)\wedge\bigwedge_{k\neq i}(\neg vpos_k(0,(j+1)mod\ m)\wedge\neg vpos_k(1,(j+2)mod\ m))\Big)\rightsquigarrow$
    $vpos_i(1,1)'\coloneqq vpos_i(0,m-1); vpos_i(0,1)'\coloneqq \bot;$
    $vpos_i(1,2)'\coloneqq vpos_i(0,m); vpos_i(0,2)'\coloneqq \bot;$
      $\vdots$
    $vpos_i(1,m)'\coloneqq vpos_i(0,m-2); vpos_i(0,m)'\coloneqq \bot$
  $leftstraight_i : \bigvee_{1\leq j\leq m}$
    $\Big(vpos_i(0,j)\wedge\bigwedge_{k\neq i}(\neg vpos_k(1,(j+1)mod\ m)\wedge\neg vpos_k(1,(j+2)mod\ m))\Big)\rightsquigarrow$
    $vpos_i(1,1)'\coloneqq vpos_i(0,m-1); vpos_i(0,1)'\coloneqq \bot;$
    $vpos_i(1,2)'\coloneqq vpos_i(0,m); vpos_i(0,2)'\coloneqq \bot;$
      $\vdots$
    $vpos_i(1,m)'\coloneqq vpos_i(0,m-2); vpos_i(0,m)'\coloneqq \bot$

Figure 7. Ambulances in the ambulance scenario.

In addition to the rules car and taxi modules have, ambulances have rules for moving two units at a time. $straightstraight_i$ moves two positions directly ahead, and the guard checks that both positions immediately ahead are unoccupied. $straightright_i$ means moving to the position which is two steps ahead in the right lane; given that the ambulance is currently in the left lane and the final position in the right lane is unoccupied. The difference between $straightright_i$ and $rightstraight_i$ is that the former also requires the position immediately ahead to be available, corresponding to driving straight and then turning right, and instead the latter also requires the position one step to the right to be available, corresponding to first turning right. Similarly for $straightleft_i$ and $leftstraight_i$.

Thus, the SRML description of the model consists of a collection of these modules. Note that the description of the modules here abstracts away from the number of vehicles in general as well as the number of the particular vehicle types. In reality, for a given number of vehicles, there are equally many modules. However, all the car (taxi, ambulance) modules are defined in the same way. For example, if there are four vehicles $\{1, 2, 3, 4\}$ and vehicles 1 and 2 are cars, vehicle 3 is a taxi, and vehicle 4 is an ambulance, then the SRML description consists of the modules named *car*-1, *car*-2, *taxi*-3 and *ambu*-4.

Normative Systems

We now go on to use SNL to describe the norms discussed above in the form of four separate normative systems (see Figure 8). Normative system η_1 has a constraint rule for each car $\{c_0, \ldots, c_q\}$ and each taxi $\{t_0, \ldots, t_r\}$, disabling any movement if they are immediately in front of an ambulance; η_2 has two constraint rules for each car, disabling switching to the right lane and moving ahead if already in the right lane; η_3 has one constraint rule for each vehicle, disabling any movement in the case that there is another vehicle immediately to the right; finally, η_4 has one constraint rule for each ambulance, disabling two-step moves in the case that there is another ambulance one step ahead in either lane.

Model Checking

For a fixed number of vehicles v, cars *cars*, taxis *taxis* and ambulances *ambu*, let $\rho(v, cars, taxis, ambu)$ denote the SRML system defined above. As noted earlier, $\rho(v, cars, taxis, ambu)$ induces a Kripke structure $\mathcal{K}_{\rho(v,cars,taxis,ambu)}$. We can now model check formulae containing references to η_1, η_2, etc., by

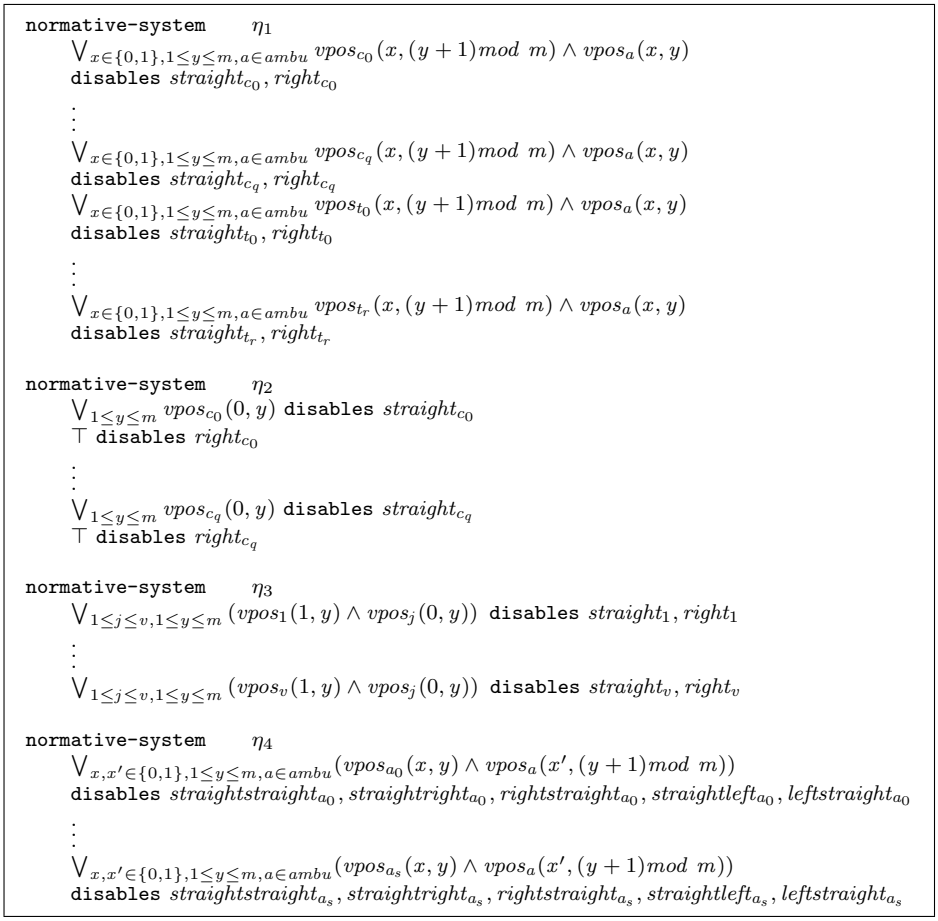

Figure 8. Normative systems η_1 to η_4.

using the SNL normative systems above as interpretations. Furthermore, we let η_{12} denote the normative system obtained by taking the union of η_1 and η_2, and so on. Let I denote the interpretation of all these normative systems.

Of primary interest is, of course, whether or not there will be crashes in the system under various circumstances. We define:

$$crash = \bigvee_{i\neq j,x\in\{0,1\},k} vpos_i(x,k)\wedge vpos_j(x,k)$$

We have the following.

(P1) Without norms, there might be crashes:

$$\mathcal{K}_{\rho(v,cars,taxis,ambu)} \models_I \mathsf{P}_{\eta_\emptyset}\Diamond crash$$

In other words, the unrestricted system permits a run along which a crash happens.

(P2) The combination of the normative systems η_1, η_2, η_3 and η_4 always ensures that there are no crashes:

$$\mathcal{K}_{\rho(v,cars,taxis,ambu)} \models_I \mathsf{O}_{\eta 1234} \Box \neg crash$$

In other words, it is an obligatory property of a run that a crash never happens.

The two properties above hold no matter how many vehicles of each type there are. The following are examples of normative properties which hold for particular configurations of vehicles:

(P3) If there are no ambulances ($ambu = \emptyset$), then η_3 ensures that there are no crashes:

$$\mathcal{K}_{\rho(v,cars,taxis,ambu)} \models_I \mathsf{O}_{\eta 3} \Box \neg crash$$

(P4) If there is only one ambulance ($ambu = \{a_0\}$), then the combination of the normative systems η_1, η_2 and η_3 ensures that there are no crashes:

$$\mathcal{K}_{\rho(v,cars,taxis,ambu)} \models_I \mathsf{O}_{\eta 123} \Box \neg crash$$

(P5) If there is more than one ambulance, then the combination of the normative systems η_1, η_2 and η_3 is not enough to ensure that there are no crashes:

$$\mathcal{K}_{\rho(v,cars,taxis,ambu)} \models_I \mathsf{P}_{\eta 123} \Diamond crash$$

A Note on Analysis

The properties above can be checked by manual inspection: this is a technically straightforward, but rather tedious process. Instead, we have analysed this case study using the MOCHA model checker [4]. MOCHA implements model checking for CTL and ATL [3] against models specified using the Reactive Systems language, of which SRML is a subset. Of course, we cannot directly check NTL and SNL properties in this way. Instead, to realise the effect of normative systems, we manually "implement" them by modifying the conditions of relevant SRML rules; this allows us to represent a (strict) subset of NTL properties as CTL formulae.

Figure 9 summarises some test results with different scenarios. A '0' under a norm means that the norm is not applied, and a '1' that it is. A

crash is possible when a '1' appears under the *Crash* column, corresponding to the formula $\mathsf{P}_\eta \Diamond crash$ being true, where η is the combination of normative systems under consideration. These test results are of course in accordance with the results presented above: property **(P1)** can be observed in rows 1, 3 and 10. Line 17 is an instance of property **(P2)**. Line 2 is an instance of property **(P3)**. Line 9 is an instance of property **(P4)**. Property **(P5)** can be observed on lines 10–16.

	#Ambulances	***#Taxis***	***#Cars***	η_1	η_2	η_3	η_4	***Crash***
1	0	0	2	0	0	0	0	1
2	0	0	2	0	0	1	0	0
3	1	0	1	0	0	0	0	1
4	1	0	1	0	0	1	0	1
5	1	0	1	0	1	1	0	1
6	1	0	1	1	0	0	0	1
7	1	0	1	1	0	1	0	1
8	1	0	1	1	1	0	0	1
9	1	0	1	1	1	1	0	0
10	2	1	1	0	0	0	0	1
11	2	1	1	0	0	1	0	1
12	2	1	1	0	1	1	0	1
13	2	1	1	1	0	0	0	1
14	2	1	1	1	0	1	0	1
15	2	1	1	1	1	0	0	1
16	2	1	1	1	1	1	0	1
17	2	1	1	1	1	1	1	0

Figure 9. Testing $\eta_1, \eta_2, \eta_3, \eta_4$ with different numbers of ambulances, taxis and cars.

6. Discussion

6.1. Related Work

The work presented in this paper has its roots in several different communities, the most significant being the tradition of using deontic logic in computer science to reason about normative behaviour of systems [30, 12, 19], and the use of model checking and temporal logics such as CTL to analyse the temporal properties of systems [14, 11].

The two main differences between the language of NTL and the language of conventional deontic logic (henceforth "deontic logic") are, first, *contextual*

deontic operators allowing a formula to refer to several different normative systems, and, second, the use of *temporal* operators. All deontic expressions in NTL refer to time: $\mathsf{P}_\eta \bigcirc \varphi$ ("it is permissible in the context of η that φ is true at the next time point"); $\mathsf{O}_\eta \Box \varphi$ ("it is obligatory in the context of η that φ always will be true"); etc. Conventional deontic logic contains no notion of time. In order to compare our temporal deontic statements with those of deontic logic we must take the temporal dimension to be implicit in the latter. Two of the perhaps most natural ways of doing that is to take "obligatory" ($\mathsf{O}\varphi$) to mean "*always* obligatory" ($\mathsf{O}_\eta \Box \varphi$), or "obligatory at the *next point in time*" ($\mathsf{O}_\eta \bigcirc \varphi$), respectively, and similarly for permission. In either case, all the principles of *Standard Deontic Logic* (SDL) (see, e.g., [9]) hold also for NTL, viz., $\mathsf{O}(\varphi \to \psi) \to (\mathsf{O}\varphi \to \mathsf{O}\psi)$ (K); $\neg \mathsf{O} \bot$ (D):[9] and from φ infer $\mathsf{O}\varphi$ (N). The two mentioned temporal interpretations of the (crucial) deontic axiom D are (both NTL validities):

$$\neg \mathsf{O}_\eta \Box \bot \text{ and } \neg \mathsf{O}_\eta \bigcirc \bot$$

A more detailed understanding of the relationship between NTL and other deontic logics would also be useful. Observe that our language is in one sense rather restricted: every deontic attitude is towards the future, never about the present or past. Indeed, when reasoning about normative behaviour of a system, it is also not easy to see what an obligation towards an objective, purely propositional formula, actually *means*. Our framework focuses on ideal *transitions*, rather than ideal *states*. This choice of design makes it also not easy to compare our set-up with other temporal deontic logics. We cannot, for instance, express properties like $\mathsf{O}_\eta \bigcirc \varphi \to \bigcirc \mathsf{O}_\eta \varphi$ (for a discussion of such "perfect recall"-like properties for temporal deontic logic, see [8]). There is another interesting direction to relax our class of formulae, however: namely, to allow for arbitrary linear temporal logic formulas $T\varphi$ in the scope of an obligation O_η.[10] This would allow us for instance to express that in system η, property φ needs to be true within three steps: $\mathsf{O}_\eta(\bigcirc \varphi \vee \bigcirc \bigcirc \varphi \vee \bigcirc \bigcirc \bigcirc \varphi)$. Semantically, this would also pave the way to define a norm as a restriction on *runs*, rather than *transitions*. One can for instance think of a norm that forbids any run in which some "unwanted" transition occurs more than n times. As a special case this would facilitate to enforce *fairness* and *liveness* conditions by a norm [15].

---

[9] Actually, the scheme D is $\mathsf{O}\varphi \to \neg \mathsf{O} \neg \varphi$, but for the logics that we consider here, both representations are equivalent.

[10] The '$\eta_\emptyset$ fragment' of this language would still be a strict subset of CTL*, so the overall language might still be well-behaved.

Contrary-to-Duty obligations are structures involving two obligations, where the second obligation "takes over" when the first is violated [21]. Logics that deal with this kind of obligation typical add actions, time, a default component or a notion of context (signalling that the primary obligation has been violated, and we are now entering a sub-ideal context) to their semantic machinery to deal with them [21]. NTL is already equipped with a temporal component, and it would certainly also be possible to label the transitions in our semantics with actions. However, given that we incorporate a suite of norms within one system, it seems NTL can also represent "sub-ideal" contexts. Technically, η_2 represents the secondary obligations that come into force when η_1 is violated, if the domain $dom(\eta_2) = \{t \mid \exists u \ (t,u) \in \eta_2\}$ is exactly the range $ran(\eta_1) = \{t \mid \exists s \ (s,t) \in \eta_1\}$ of η_1. We leave a detailed comparison between existing temporal deontic logics and NTL for future works, as well as any investigation into the usefulness of NTL to model contrary-to-duty obligations.

It has been argued that "*deadlines* are important norms in most interactions between agents" [13, page 40], and this naturally suggests the need for a temporal component in reasoning about systems with norms. Indeed, the authors of [13] used CTL in their paper *Designing a Deontic Logic of Deadlines* [7], and one of their authors reduces Strategic Deontic Temporal Logic to ATL in [6][11].

One of our concerns in this paper was to give a *computationally grounded* semantics for deontic modalities, in that we aim to give the semantics a clear computational interpretation; in this respect, our work is similar in spirit to the *deontic interpreted systems* model of Lomuscio and Sergot [19]. Perhaps the most obvious difference is that while we consider "bad transitions", Lomuscio and Sergot are concerned with "bad states".

We should also mention work by Sergot and Craven on the use of variants of the C+ language for representing and reasoning about deontic systems [23, 24]. The nC+ language they develop can be understood as an alternative to SRML/SNL for defining (symbolic) representations of Kripke models and normative systems. The main difference is that the C+ language provides a richer, higher level, arguably more general, logical framework for specifying models than SRML/SNL. In fact their work emerges from a rather different community — reasoning about action and non-monotonic reasoning in artificial intelligence. It would be interesting to undertake a more formal investigation of the relationship between the two frameworks, with respect to both expressive power and computational complexity. It seems

[11] The latter paper includes an application to Chisholm's paradox.

plausible that analogues of our SRML/SNL model checking problems will be more complex in the richer framework of nC+, although we emphasise that currently we have no results here. Note that the first of the papers cited above also presents an outline of how normative properties expressed in CTL can be evaluated using standard model checking tools, and uses an example similar to that used in this paper.

Finally, NTL is closely related to a system called Normative ATL, which was introduced in [31]. In fact, NTL is related to CTL [14] in the same way as that Normative ATL is related to ATL [3]: however, NTL (and specifically its semantics) is much simpler (and we believe more intuitive) than Normative ATL [31], and we present many more technical results associated with our logic.

6.2. Future Work

A number of issues suggest themselves for future work:

- Regarding NTL, tight bounds for complexity of uninterpreted symbolic model checking, and the complexity of satisfiability, which has not been addressed within this paper.
- The calculus of normative systems, as mentioned in Section 2, could be developed further. In this paper, we have considered only set-theoretic operations on normative systems, (taking their union and intersection), but other possible operators might be considered as well, such as what happens when a normative system is "restricted" to some set of players, or when it is restricted to those constraints that some players regard as in their best interests. To capture these latter concerns, we would need some notion of preference or goals.
- Another issue of interest is that of "reasonableness", and in particular the extent to which this constraint is necessary.
- We might usefully consider the possibility and implications of *non-compliance*. It seems inevitable that in real systems, some agents will fail to comply with a normative system, even if it is in their best interests to do so. This raises two issues: First, what is the right way to go about dealing with this possibility with respect to the design of normative systems themselves, and second, how are we to deal with these concerns at the language level?
- We might also consider prioritised collections of normative systems ("if this normative system fails, then use this").

- Finally, of course, a full implementation of a model checker encompassing the variations discussed in Section 4 would be desirable, and as ever, more detailed case studies would be useful to further evaluate the logic.

6.3. Conclusions

The design and application of normative systems and social laws is a major area of research activity in the multi-agent systems community. If we are going to make use of such social laws, then it seems only appropriate that we develop formalisms that allow us to explicitly and directly reason about them. In this paper, we have developed a Normative Temporal Logic that is intended for this purpose, being careful to give a semantics to deontic modalities that has a clear computational interpretation. We see the key advantages of NTL as follows. First, the fact that the formalism is so closely related to CTL is likely to be an advantage from the point of view of comprehension and acceptance within the mainstream model checking/verification community. Second, the fact that the language has a clear computational interpretation means that it can be applied in a computational setting without any ambiguity of interpretation. Third, the clear identification of different normative systems within the language, and the ability to talk about these directly, represents a novel step forward. While NTL arguably lacks some of the nuances of more conventional deontic and deontic temporal logics, we believe these advantages imply that the language and the approach it embodies merit further research.

Acknowledgements. We would like to thank the reviewers for their helpful comments and suggestions. This work was partially funded by projects AT (CONSOLIDER CSD2007-0022), IEA (TIN2006-15662-C02-01) and 2006 5 OI 099.

References

[1] Alur, R., and T. Henzinger, 'Computer-aided verification', 1999. Manuscript.

[2] Alur, R., and T. A. Henzinger, 'Reactive modules', *Formal Methods in System Design*, 15 (11): 7–48, 1999.

[3] Alur, R., T. A. Henzinger, and O. Kupferman, 'Alternating-time temporal logic', *Journal of the ACM*, 49 (5): 672–713, 2002.

[4] Alur, R., T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran, 'Mocha: Modularity in model checking', in *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, Springer-Verlag, Berlin, Germany, 1998, pp. 521–525.

[5] Attie, Paul C., and E. Allen Emerson, 'Synthesis of concurrent systems with many similar processes', *ACM Transactions on Programming Languages and Systems*, 20 (1): 51–115, 1998.

[6] Broersen, J., 'Strategic deontic temporal logic as a reduction to ATL, with an application to Chisholm's scenario', in *Proceedings Eighth International Workshop on Deontic Logic in Computer Science (DEON'06)*, vol. 4048 of *LNAI*, Springer, 2006, pp. 53–68.

[7] Broersen, J., F. Dignum, V. Dignum, and J.-J.Ch. Meyer, 'Designing a deontic logic of deadlines', in A. Lomuscio, and D. Nute (eds.), *Proceedings Seventh International Workshop on Deontic Logic in Computer Science (DEON'04)*, vol. 3065 of *LNAI*, Springer, 2004, pp. 43–56.

[8] Brunel, J., *Combining Temporal and Deontic Logics. With an Application to Computer Security*, Ph.D. thesis, IRIT, Toulouse, France, 2007.

[9] Carmo, Jose, and Andrew J. I. Jones, 'Deontic logic and contrary-to-duties', in D. M. Gabbay, and F. Guenthner (eds.), *Handbook of Philosophical Logic,* 2^{nd} *edition*, vol. 8, Kluwer Academic Publishers, 2002, pp. 265–343.

[10] Cheng, A., 'Complexity results for model checking', Tech. Rep. RS-95-18, BRICS, Department of Computer Science, University of Aarhus, 1995.

[11] Clarke, E. M., O. Grumberg, and D. A. Peled, *Model Checking*, The MIT Press, Cambridge, MA, 2000.

[12] Dignum, F., 'Autonomous agents with norms', *Artificial Intelligence and Law*, 7: 69–79, 1999.

[13] Dignum, F., J. Broersen, V. Dignum, and J.-J.Ch. Meyer, 'Meeting the deadline: Whey, when and how', in M.G. Hinchey, J.L. Rash, W.F. Truszkowski, and C.A. Rouff (eds.), *Formal Approaches to Agent-Based Systems*, vol. 3228 of *LNAI*, Springer, 2004, pp. 30–40.

[14] Emerson, E. A., 'Temporal and modal logic', in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990, pp. 996–1072.

[15] Francez, N., *Fairness*, Springer-Verlag, Berlin, Germany, 1986.

[16] Harel, D., D. Kozen, and J. Tiuryn, *Dynamic Logic*, The MIT Press, Cambridge, MA, 2000.

[17] Hoek, W. van der, A. Lomuscio, and M. Wooldridge, 'On the complexity of practical ATL model checking', in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, Hakodate, Japan, 2005, pp. 201–208.

[18] Hoek, W. van der, M. Roberts, and M. Wooldridge, 'Social laws in alternating time: Effectiveness, feasibility, and synthesis', *Synthese*, 156 (1): 1–19, 2007.

[19] Lomuscio, A., and M. Sergot, 'Deontic interpreted systems', *Studia Logica*, 75 (1): 63–92, 2003.

[20] Moses, Y., and M. Tennenholtz, 'Artificial social systems', *Computers and AI*, 14 (6): 533–562, 1995.

[21] Prakken, H., and M. Sergot, 'Contrary-to-duty obligations', *Studia Logica*, 57 (1): 91–115, 1996.

[22] Schnoebelen, P., 'The complexity of temporal logic model checking', in P. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyascev (eds.), *Advances in Modal Logic Volume 4*, King's College Publications, London, 2003, pp. 393–436.

[23] Sergot, M., 'Modelling unreliable and untrustworthy agent behaviour', in B. Dunin-Keplicz, A Jankowski, A. Skowron, and M. Szczuka (eds.), *Monitoring, Security, and Rescue Techniques in Multiagent Systems*, Advances in Soft Computing, Springer, 2005, pp. 161–178.

[24] Sergot, M., and R. Craven, 'The deontic component of action language nC+', in L. Goble, and J.-J. Ch. Meyer (eds.), *Deontic Logic and Artificial Systems. Proc. 8th International Workshop on Deontic Logic in Computer Science*, no. 4048 in LNAI, Springer, 2006, pp. 222–237.

[25] Shoham, Y., and M. Tennenholtz, 'Emergent conventions in multi-agent systems', in C. Rich, W. Swartout, and B. Nebel (eds.), *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, 1992, pp. 225–231.

[26] Shoham, Y., and M. Tennenholtz, 'On the synthesis of useful social laws for artificial agent societies', in *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Diego, CA, 1992, pp. 276–281.

[27] Shoham, Y., and M. Tennenholtz, 'On social laws for artificial agent societies: Off-line design', in P. E. Agre, and S. J. Rosenschein (eds.), *Computational Theories of Interaction and Agency*, The MIT Press, Cambridge, MA, 1996, pp. 597–618.

[28] Shoham, Y., and M. Tennenholtz, 'On the emergence of social conventions: Modelling, analysis, and simulations', *Artificial Intelligence*, 94 (1-2): 139–166, 1997.

[29] Stockmeyer, L. J., and A. K. Chandra, 'Provably difficult combinatorial games', *SIAM Journal of Computing*, 8 (2): 151–174, 1979.

[30] Wieringa, R. J., and J.-J. Ch. Meyer, 'Deontic logic in computer science', in J.-J. Ch. Meyer, and R. J. Wieringa (eds.), *Deontic Logic in Computer Science — Normative System Specification*, John Wiley & Sons, 1993, pp. 17–40.

[31] Wooldridge, M., and W. van der Hoek, 'On obligations and normative ability: Towards a logical analysis of the social contract', *Journal of Applied Logic*, 4 (3-4): 396–420, 2005.

Thomas Ågotnes
Bergen University College
PO Box 7030
N-5020 Bergen
Norway
tag@hib.no

Juan A. Rodríguez-Aguilar
& Carles Sierra
CSIC-IIIA
University
Address
Spain
{jar,carles}@iiia.csic.es

Wiebe van der Hoek
& Michael Wooldridge
Department of Computer Science
University of Liverpool
Liverpool
L69 3BX United Kingdom
{wiebe,mjw}@csc.liv.ac.uk