Practical Run-Time Norm Enforcement with Bounded Lookahead

Natasha Alechina
University of Nottingham
Nottingham, UK
nza@cs.nott.ac.uk

Nils Bulling
Delft University of Technology
Delft, The Netherlands
n.bulling@tudelft.nl

Mehdi Dastani
Universiteit Utrecht
Utrecht, The Netherlands
M.M.Dastani@uu.nl

Brian Logan
University of Nottingham
Nottingham, UK
bsl@cs.nott.ac.uk

ABSTRACT

Norms have been widely proposed as a means of coordinating and controlling the behaviour of agents in a multi-agent system. A key challenge in normative MAS is norm enforcement: how and when to restrict the agents' behaviour in order to obtain a desirable outcome? Even if a norm can be enforced theoretically, it may not be enforceable in a grounded, practical setting. In this paper we study the problem of practical norm enforcement. The key notion is that of a *guard*. Guards are functions which restrict the possible actions after a history of events. We propose a formal, computational model of norms, guards and norm enforcement, based on linear-time temporal logic with past operators. We show that not all norms can be enforced by such guard functions, even in the presence of unlimited computational power to reason about future events. We analyse which norms can be enforced by guards if only a fixed lookahead is available. We investigate decision problems for this question with respect to specific classes of norms, related to safety and liveness properties.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multi-agent systems*

Keywords

Norms; Run-time enforcement

1. INTRODUCTION

Multi-agent systems are often regulated using norms. Some norms can be violated (agents are able to perform an action which leads to a violation) and the violating behaviour is sanctioned. Some norms are regimented, that is, agents are prevented from performing an action that would lead to a violation. In this paper we focus on regimented norms, and in particular on the case when norms are temporal properties which potentially apply to infinite runs of the system (the simplest case of such a norm is an invariant property to be maintained). In a practical system, regimented norm enforcement involves having access to a finite run of the system, and having to decide which successor states should be disallowed since all the subsequent runs through those states would result in a norm violation.

We assume that the decision regarding which successor states should be enabled is provided by a *guard* function, which takes a finite run (history of the system's behaviour so far) and returns 'safe' successor states: the states from where it is possible to continue without violating the norm. The central problem in this setting is defining an appropriate guard function. The guard function should have several properties: (1) it must ensure that the resulting runs do not violate the norm; (2) it must not rule out behaviours that do not violate the norm; (3) it should be deadlock-free, that is every compliant history should have at least one successor state. Note that the problem of specifying the guard function is different from that of run-time verification [9], where the problem is to check whether a finite run violates a given property (rather than prevent a next step on the run if it leads to a violation).

Clearly, achieving the three properties above is not always possible. Not all norms can be perfectly enforced using a guard function (perfect enforcement means that the set of all infinite runs generated by the guard function is exactly the set of runs that conform to the norm). An obvious example of such norms are liveness properties (or, in normative terms, obligations without a deadline). Similarly, deadlock-free guard functions do not always exist: in the simplest case, a norm may not be enforceable on a given system.

We assume that the normative organisation has access to a finite transition system describing all possible multi-agent system behaviours, but only a bounded lookahead window that can be used to decide which successor states of the current state on a run should be disallowed. We study the minimal size of a lookahead window required for successful norm enforcement and show that it is surprisingly large for norms expressed in LTL.

The contributions of the paper are as follows.

- We define the notions of run-time norm enforcement, guard functions and the problem of establishing whether a given norm has a deadlock-free guard that (perfectly) enforces it on a given transition system.
- We study a special kind of guard function that correspond to LTL with past formulas, and characterise which norms can be perfectly enforced by LTL with past formulas and the computational complexity of checking whether an LTL with past formula is a deadlock-free guard.
- We show that liveness norms can always be enforced and

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

safety norms can be perfectly enforced (in the sense of Definition 10 below) by the *canonical guard* with a sufficiently large lookahead.

- We show that state-based safety norms can be always enforced with a lookahead window of size 1 by explicitly listing 'prohibited' successor states; however some enforceable LTL norms cannot be enforced with a lookahead window that is smaller than the length of the longest cycle-free path is a model.

The structure of the paper is as follows. We first define the system model and LTL with past operators. Then, we introduce the three key concepts of this paper: norms, guards and norm enforcement, followed by a logic-based setting to define guards. In Section 5, we consider bounded lookahead, and discuss practical aspects in Section 6. Finally, we discuss related work in Section 7 and conclude.

2. PRELIMINARIES

System Model. Our system model is a transition system consisting of a set of states and a serial accessibility relation between states. Each state is labelled with a set of propositions, modelling the facts in the world.

DEFINITION 1 (TRANSITION SYSTEM). *Let Π be a finite set of atomic propositions. A (transition) system is defined as $M = \langle S, R, V\rangle$, where S is a set of states, $R \subseteq S \times S$ is a serial accessibility relation and $V : S \to 2^{\Pi}$ is a labelling function. In order to simplify the presentation, we sometimes implicitly assume that there is a non-empty set $S_0 \subseteq S$ of* initial states*. If not stated otherwise, it can be assumed that $S_0 = S$.*

A transition system M describes the computational behaviour of a (multi) agent system. The relation R corresponds to the actions agents may execute, sequentially or in parallel, in each state to bring the system to a new state. A *history* and a *run* in the transition system are finite and infinite sequences of states interconnected by the accessibility relation, respectively.

DEFINITION 2 (RUNS AND HISTORIES). *Given a transition system $M = \langle S, R, V\rangle$, a* run *of M is an infinite sequence of states $\rho = s_0 s_1 s_2 \ldots$ such that for $i \geq 0$ we have $(s_i, s_{i+1}) \in R$. The set of* runs *of M starting in s is denoted by $\mathcal{R}_M(s)$. We simply write $\mathcal{R}_M := \bigcup_{s\in S} \mathcal{R}_M(s)$ for the set of all runs, and $\mathcal{R}_M(S_0)$ as the set of runs starting in S_0. A* history *h of M is a finite prefix of a run. We use $h \sqsubset \rho$ to indicate that h is a (prefix) history of the run ρ; $h \sqsubset h'$ to indicate that the history h is a prefix of the history h'; and $h \sqsubseteq h'$ if $h \sqsubset h'$ or $h = h'$. The set of histories of M starting in s is denoted by $\mathcal{H}_M(s)$ and, analogously to runs, we define $\mathcal{H}_M$ and $\mathcal{H}_M(S_0)$. Finally, the operator $\circ$ is used to concatenate a history h with another history h' ($h \circ h'$) and run ρ ($h \circ \rho$), respectively. We often omit $\circ$ and just write hs, hh', $h\rho$, etc.*

Given a run ρ we write $\rho[i]$, $\rho[i, j]$, $\rho[i, \infty]$, for $i \geq 0$, to refer to the ith state on ρ, the history starting at position i on ρ and ending at position j of ρ, and the suffix (run) of ρ starting at position i on ρ, respectively. We assume that i and j are non-negative integers with $i \leq j$. We use similar notation for histories; in particular, for a history h, $h[\infty]$ refers to the *last state of h*. We omit the formal details.

LTL with past operators. Below, we use LTL to specify norms and LTL with past operators [23] to specify guards: the temporal operator Y stands for "yesterday", P for "now or at some point in the past", S for "since", X for "tomorrow (next)", F for "now or sometime in the future", and U for "until". We denote LTL with past operators by LTL^{+p}.

DEFINITION 3 (LTL PLUS PAST, SIMPLE FORMULA). *Formulae of LTL^{+p} are defined by the grammar: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid Y\phi \mid \phi S\phi \mid X\phi \mid \phi U\phi$ where $p \in \Pi$ is a proposition. We define $F\phi = \top U\phi$ and $P\phi = \top S\phi$ as macros. Given a transition system M, a run ρ, and a position $i \in \mathbb{N}_0$, the truth of LTL^{+p}-formulas is defied by the semantic relation $\models$ as follows:*

- $M, \rho, i \models p$ *iff* $p \in V(\rho[i])$
- $M, \rho, i \models \neg\phi$ *iff* $M, \rho, i \not\models \phi$
- $M, \rho, i \models \phi \wedge \psi$ *iff* $M, \rho, i \models \phi$ *and* $M, \rho, i \models \phi$
- $M, \rho, i \models Y\phi$ *iff* $i > 0$ *and* $M, \rho, i-1 \models \phi$,
- $M, \rho, i \models \phi S\psi$ *iff* $\exists j \leq i$ *s.t.* $M, \rho, j \models \psi$ *and* $\forall k : j < k \leq i : M, \rho, k \models \phi$
- $M, \rho, i \models X\phi$ *iff* $M, \rho, i+1 \models \phi$
- $M, \rho, i \models \phi U\psi$ *iff* $\exists j, i \leq j$ *s.t.* $M, \rho, j \models \psi$ *and* $\forall k : i \leq k < j : M, \rho, k \models \phi$

LTL-formulae are LTL^{+p}-formulae that do not contain any past-time modalities, i.e. Y, P and S. We call an LTL^{+p}-formula simple *if it contains at most one temporal operator.*

We note that LTL^{+p} over infinite runs is no more expressive than pure LTL (without past operators). However, the past operators allow more intuitive and succinct specifications [20].

3. NORMS, GUARDS AND ENFORCEMENT

In this section, we introduce the three key concepts of this paper: norms, guards and norm enforcement. A *norm* can be seen as a linear-time property over a set of propositions, or as a set of 'good' runs of a transition system uniquely defining a linear time property. A *guard* is a means to control the system's run-time behaviour; it allows to disable transitions after a history. Finally, the notion of *enforcement* links norms and guards. We say that a guard enforces a norm if all possible behaviours resulting from the application of the guard are normative. Throughout this section we assume that $M = (S, R, V)$ is a finite transition system with initial states $S_0 \subseteq S$ over a finite set of propositions Π. We also assume that formulae are constructed over Π.

3.1 Specification of Regimentation Norms

We begin with our representation of norms. A *norm* describes the 'good', the *normative* behaviour of a system. In general, this behaviour can be defined independently of a transition system, just over a set of propositions Π. In this sense, a norm is a set of infinite words over 2^{Π}. For example, the norm "it is prohibited to take drugs" corresponds to the set of all infinite words $w_0 w_1 \ldots \in 2^{\Pi}$ such that $drugs \notin w_i$ for all $i \in \mathbb{N}_0$. If a norm is considered in the context of a transition system it is often convenient to identify a linear-time properties with a subset of the runs of the transition system, namely those runs which (given the labelling function V) correspond to one of the behaviours of the norm. In the remainder of this paper we use LTL-formulae φ to characterise a norm. This supports both views: if we talk about a norm independent of a model, the (LTL-)norm φ is identified with all linear-time behaviours satisfying φ (over a fixed set of proposition). In the context of a transition system M, we write φ_M to refer to a norm defined as the set of all runs $\rho \in \mathcal{R}_M(S_0)$ that satisfy the LTL-formula φ. The formal definition is given next.

DEFINITION 4 (REGIMENTATION NORM). *A (regimentation)* norm *is an LTL-formula φ. A* norm φ over a transition system M *is given by*

$$\varphi_M := \{\rho \in \mathcal{R}_M(S_0) \mid M, \rho, 0 \models \varphi\}.$$

As seen from this definition a norm has a descriptive flavour. It specifies, from a global point of view, which runs or linear time behaviours are good.

3.2 Guards

A transition system encodes all possible behaviours, both the 'good' ones as well as the 'bad'/undesirable ones according to a given norm. The system designer may wish to control the behaviour of the agents to ensure a desirable system outcome. Using the notation introduced above, the system designer needs a mechanism to ensure that all behaviours are normative, but which at the same time does not overly restrict the agents' autonomy. *Guards* are used to implement norms by disabling specific transitions following a history. It is important to note that guards operate and restrict behaviours at *run-time*. This is in general more flexible than approaches operating at *design-time*, in the sense that transitions in the (static) model are altered. Moreover, a run-time approach is in general less restrictive and offers agents more autonomy as the disabling of transitions dynamically depends on the history.

A *guard over a transition system M* is a function that maps histories to a set of states enabled after the history.

DEFINITION 5 (GUARD). *A guard over M is defined as $G_M : \mathcal{H}_M \to 2^S$.*

If the system restricts the actions according to the guard some histories are no longer possible. A history that is possible given the application of a guard is called *consistent (with the guard)*.

DEFINITION 6 (CONSISTENT HISTORY). *Let G_M be a guard and $h \in \mathcal{H}_M$. We call a history h* consistent with the guard *G_M if, and only if, $h = s \in S_0$ or $s \in G_M(h')$ for all $h' \in \mathcal{H}_M$, $s \in S$ with $h' \circ s \sqsubset h$.*

A guard is deadlock free if each consistent history can be extended to another consistent history.

DEFINITION 7 (DEADLOCK-FREE). *A guard G_M is called* deadlock-free*, or* df-guard *for short, if, and only if, $G_M(h) \neq \emptyset$ for all histories $h \in \mathcal{H}_M(S_0)$ that are consistent with G_M.*

The application of a guard restricts the set of runs of a transitions system as follows:

DEFINITION 8 (GUARD APPLICATION). *The* application of a df-guard G_M in *s restricts the runs starting in s to*

$$\mathcal{R}_M^G(s) = \{\rho \in \mathcal{R}_M(s) \mid \forall h \in \mathcal{H}_M, s' \in S, hs' \sqsubset \rho : s' \in G_M(h)\}.$$

It is easy to see that if G_M is a df-guard then $R_M(s) \neq \emptyset$ for all $s \in S_0$. We note that the transitions disabled by the guard are not disabled in the model but at run-time.

Clearly, a more general guard is preferable as it allows the agents greater autonomy. This is captured by the following definition.

DEFINITION 9 (GENERALITY ORDERING). *Given two guards G_M and G'_M we say that G'_M is at least as general as G_M, denoted by $G_M \leq G'_M$, if, for all $h \in \mathcal{H}_M$, $G_M(h) \subseteq G'_M(h)$. As usual, we use $<$ to refer to the strict variant.*

The following proposition captures the intuition that the application of a more general guard results in a larger set of system behaviours.

PROPOSITION 1. *If $G_M \leq G'_M$ for two df-guards, then $\mathcal{R}_M^G(s) \subseteq \mathcal{R}_M^{G'}(s)$ for all $s \in S_0$.*

3.3 Norm Enforcement

A key question is whether we can construct a guard such that, if the guard is applied, then all possible runs of the system are normative. This is formally captured by the following definition:

DEFINITION 10 ((PERFECT, OPTIMAL) NORM ENFORCEMENT). *We say that a df-guard G_M enforces a norm φ_M iff $\varphi_M = \emptyset$ or $\emptyset \neq \mathcal{R}_M^G(S_0) \subseteq \varphi_M$. The enforcement is* perfect *iff $\mathcal{R}_M^G(S_0) = \varphi_M$. The enforcement is* optimal *if there is no other df-guard G'_M such that $\mathcal{R}_M^G(S_0) \subsetneq \mathcal{R}_M^{G'}(S_0) \subseteq \varphi_M$.*

EXAMPLE 1 (A LIVENESS NORM). *Consider the transition system $M_0 = \langle S, R, V\rangle$ with $S = \{s_0, s_1\}$, $R = \{(s_0, s_0), (s_0, s_1), (s_1, s_1)\}$, and $V(s_0) = \emptyset$, $V(s_1) = \{p\}$, and $S_0 = S$. Possible runs in the transition system are: $s_0^\omega = s_0 s_0 \ldots$ (always stay in s_0), $s_0 s_1^\omega = s_0 s_1 s_1 \ldots$ (stay in s_0 once, then proceed to s_1 and stay there forever), $s_0^2 s_1^\omega = s_0 s_0 s_1^\omega$ (stay in s_0 twice, then proceed to s_1 and stay there forever), and s_1^ω (stay in s_1 forever). We consider the norm $\varphi_0 = Fp$, the set of all runs which eventually visit a state in which proposition p holds, formally: $(\varphi_0)_{M_0} = \{s_0^n s_1^\omega \mid n \in \mathbb{N}_0\}$. Intuitively, the norm states that it is prohibited to stay in s_0 forever. This is a* liveness property. *We would like to construct a guard which enforces the norm. We define the following guard function G_{1,M_0}: $G_{1,M_0}(h) = \{s_1\}$ for all histories h. It is easy to see that this norm enforces $(\varphi_0)_{M_0}$ as we have: $\mathcal{R}_{M_0}^{G_1} = \{s_0 s_1^\omega, s_1^\omega\} \subseteq (\varphi_0)_{M_0}$. Clearly, the guard does not perfectly nor optimally enforce $(\varphi_0)_{M_0}$ as for example $s_0 s_0 s_1^\omega \notin \mathcal{R}_{M_0}^{G_1}$ but the run is normative. Another alternative would be guard function G_{2,M_0} with $G_{2,M_0}(h) = \{s_0, s_1\}$ if h ends in s_0, and $G_{2,M_0}(h) = \{s_1\}$ ends in s_1. This norm, however, does not enforce $(\varphi_0)_{M_0}$. Actually, there is no guard function which can perfectly enforce $(\varphi_0)_{M_0}$. We will make this more formal in Section 4.2.*

EXAMPLE 2 (A SAFETY NORM). *Consider the transition system $M_1 = (S_1, R_1, V_1)$ with $S_1 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ with $R_1 = \{(s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_3), (s_0, s_4), (s_4, s_5), (s_5, s_5)\}$ and $V(s) = \{p\}$ for all $s \in S_1 \backslash \{s_3\}$ and $V(s_3) = \emptyset$. The norm $\varphi_1 = Gp$ with $(\varphi_1)_{M_1} = \{s_0 s_4 s_5^\omega\}$ is a safety property. The guard G_{3,M_1} with $G_{3,M_1}(s_0) = \{s_4\}$, $G_{3,M_1}(s_0 s_4) = \{s_5\}$, and $G_{3,M_1}(s_0 s_4 s_5^n) = \{s_5\}$ for all $n \in \mathbb{N}$ does perfectly enforce $(\varphi_1)_{M_1}$ for $S_0 = \{s_0\}$. In Section 6.2 we show that all safety properties can be perfectly enforced.*

4. LOGIC-BASED GUARDS

In this section we define guards in LTL^{+p} and introduce the important concept of a *canonical guard*. We investigate when guards are deadlock free and when norms can be perfectly enforced. As before, we assume that $M = (S, R, V)$ is a finite transition system with initial states S_0 over a finite set of propositions Π.

4.1 Guards in LTL+Past

We first show how LTL^{+p}-formulae can be used to define guards. The guard G_M^γ, defined by the *guard formula* γ, returns, for a history h, the set of states s such that, on some run ρ that extends the history hs, γ holds.

DEFINITION 11 (LTL^{+p}-DEFINED GUARD). *For a given LTL^{+p}-formula γ, we define the γ-*defined guard *over M G_M^γ by*

$$G_M^\gamma(h) = \{s \in S \mid \exists \rho \in \mathcal{R}_M : hs \sqsubset \rho \text{ and } M, \rho, |h| \models \gamma\}.$$

We sometimes refer to γ as the guard formula*. When G_M^γ is applied to M we simply write $\mathcal{R}_M^\gamma$ instead of $\mathcal{R}_M^{G^\gamma}$ to refer to the set of possible runs when the guard G_M^γ is applied.*

It is important to note that, in addition to the current history, such a guard can make use of additional information contained in the model. That is, the decision which transitions to disable can take into account the future and the past. In this sense the guard has *unbounded lookahead*. In Section 5 we consider guards with *bounded lookahead*.

REMARK 1. *We note that the guard γ is interpreted in/after the history hs. Equivalently, it would be possible to interpret it in history h.*

Deadlock-free guards are particularly important. According to the definition of a df-guard, the set of states returned must always be non-empty; that is, it must always be possible for the system to continue execution following a history. In general, this is not ensured for guards defined using guard formulas; consider, for example, a formula that is never true in the model. So, a key problem is to check whether a guard formula does actually define a df-guard. In Section 4.3 we show that this is a difficult problem.

EXAMPLE 3 (EXAMPLE 1 CONT.). *We consider the guard $G^{\gamma_1}_M$ with $\gamma_1 = p$. We have that $G^{p}_{M_0} = G_{1,M_0}$. We can also define G_{2,M_0} by $G^{\top}_{M_0}$. We consider the guard γ_{φ_0} (this guard will play an important role in the remainder of the paper and is formally introduce in Definition 12). The guard $G^{\gamma_{\varphi_0}}_{M_0}$ with $\gamma_{\varphi_0} = P(Y\bot \wedge \varphi_0)$ seems to be a good choice to enforce φ_{M_0}. Intuitively, it enables all paths that satisfy the norm. But the guard does not enforce $(\varphi_0)_{M_0}$, since it allows the run s_0^ω. In general, if a norm corresponds to a property that is not violated on any finite initial segment of a run, then it is not possible to perfectly enforce it using guards. This relates to run-time verification and the non-existence of finite bad prefixes for liveness properties [9].*

EXAMPLE 4 (EXAMPLE 2 CONT.). *The guard $G^{\gamma_3}_{M_1}$ with $\gamma_3 = Gp$ is equivalent to G_{3,M_1}. In this example, the* canonical guard *$\gamma_{\varphi_1} = P(Y\bot \wedge Gp)$ does enforce $(\varphi_1)_{M_1}$, perfectly and thus also optimally.*

4.2 Canonical Guards

Given a norm φ it is possible to define a specific guard from the norm itself: the *canonical guard*. We have already seen this guard in Examples 3 and 5. If the canonical guard enforces a norm then it is the most general guard which enforces the norm. The canonical guard has other interesting properties which we discuss below.

DEFINITION 12 (CANONICAL GUARD). *Let φ be a norm. The* canonical guard (formula) *of φ is defined as $\gamma_\varphi = P(Y\bot \wedge \varphi)$.*

Before studying properties of the canonical guard we make the following easy observation.

PROPOSITION 2. *If $\models \gamma \rightarrow \gamma'$ then for any transition system M, $G^{\gamma}_M \leq G^{\gamma'}_M$.*

The following result states that if the canonical guard of a norm is deadlock-free and enforces the norm, then it is the optimal df-guard in the sense that it leaves the agents with the greatest degree of autonomy.

PROPOSITION 3. *If the canonical guard $G^{\gamma_\varphi}_M$ of a norm φ is deadlock-free and enforces φ_M then it is optimal.*

PROOF. Consider a df-guard G^{γ}_M that enforces φ_M. That is, for all $\rho \in \mathcal{R}^{\gamma}_M$ we have that $\rho \in \varphi_M$ which means that $M, \rho, 0 \models \rho$. So, if $G^{\gamma}_M(h) \neq \emptyset$ there is a run ρ with $h \sqsubset \rho$ and $\rho \in \mathcal{R}^{\gamma}_M$. This show that $M, \rho, |h| \models \gamma_\varphi$ and thus $G^{\gamma_\varphi}_M(h) \neq \emptyset$. We conclude that $G^{\gamma}_M \leq G^{\gamma_\varphi}_M$. □

The canonical guard can also be used to show that there is no df-guard to enforce a norm from a given history. In other words, Proposition 3 tells us that, in order to enforce a norm from a history h with $G^{\gamma_\varphi}_M(h) = \emptyset$, we must define a df-guard which regiments the history h.

PROPOSITION 4. *Let φ be a norm. If there is a history $h \in \mathcal{H}_M$ with $G^{\gamma_\varphi}_M(h) = \emptyset$ then there is no df-guard which can enforce φ from history h on.*

PROOF. Suppose $G^{\gamma_\varphi}_M(h) = \emptyset$ and there were a df-guard G^{γ}_M with $G^{\gamma}_M(h) \neq \emptyset$ and for all runs $\rho \in \mathcal{R}^{\gamma}_M$ with $h \sqsubset \rho$ we have that $\rho \in \varphi_M$. Then, we must have that $M, \rho, |h| \models \gamma_\varphi$ and thus $\rho[|h|] \in G^{\gamma_\varphi}_M(h) = \emptyset$. Contradiction. □

Finally, the canonical guard suggests a decision method for determining whether a norm can be enforced perfectly: a norm cannot be perfectly enforced if the canonical guard does not enforce the norm.

PROPOSITION 5. *Given a norm φ. If $G^{\gamma_\varphi}_M$ does not enforce φ_M then φ_M cannot be perfectly enforced by any guard.*

PROOF. Suppose $\rho^* \in \mathcal{R}^{\gamma_\varphi}_M \backslash \varphi_M$ is a run witnessing that the canonical guard does not enforce the norm. Because $\rho^* \in \mathcal{R}^{\gamma_\varphi}_M$ we have that for any $h \sqsubset \rho^*$, $G^{\gamma_\varphi}_M(h) \neq \emptyset$. In particular, for any such h there must be a run ρ_h with $h \sqsubset \rho_h$ and $\rho_h \in \varphi_M$ because the guard is canonical. Now, suppose there were a guard G^{γ}_M that enforces φ_M. Then, it cannot be the case that $\rho^* \in \mathcal{R}^{\gamma}_M$. To avoid this, the guard has to regiment from some h the transition which results in ρ^*. Suppose h denotes this very history where the regimentation takes place and $hs \sqsubset \rho^*$. Then, however, the run $\rho_{hs} \in \varphi_M$ (using the notation above) would also not be in $\mathcal{R}^{\gamma}_M$. This shows that G^{γ}_M cannot perfectly enforce φ_M. □

4.3 Determining Deadlock Freeness

Many of these results rely on deadlock-free guards. Deadlock-free guards are also of practical importance, as it is often undesirable if a system deadlocks. Unfortunately, a LTL^{+p}-defined guard is not necessarily deadlock free. As a result, using a guard to regiment the behaviour of the system can result in deadlock. The following proposition suggests that, in general, verifying whether a guard is deadlock-free is a difficult problem. However, we can identify classes of guards with better computational properties.

PROPOSITION 6. *The problem of checking whether G^{γ}_M is deadlock-free is in **2EXPTIME**. If γ is simple (cf. Definition 3) and contains no future time operators, then it is in **PSPACE**, and if γ is a Boolean formula, then in **P**.*

PROOF. We have that G^{γ}_M is deadlock free, iff, for all $s \in S_0$ we have that $M, s, 0 \models EX\gamma \wedge AG(AX\neg\gamma \rightarrow P\neg\gamma)$. For an arbitrary LTL^{+p}-formula γ, this is a formula of $\mathsf{CTL}^* + \mathsf{past}$, and the model checking problem for $\mathsf{CTL}^* + \mathsf{past}$ is in **2EXPTIME** [10]. If γ is simple (cf. Definition 3) and contains no future time operators, then the formula above is equivalent to $EXE\gamma \wedge AG(AXA\neg\gamma \rightarrow EPE\neg\gamma)$ (where we assume linear past) which is a $\mathsf{CTL} + \mathsf{past}$ formula, and the model checking problem for $\mathsf{CTL} + \mathsf{past}$ is **PSPACE**-complete [19]. If γ is a Boolean formula, then the initial formula is equivalent to the future time CTL-formula $EX\gamma \wedge \neg E(\gamma U(\gamma \wedge AX\neg\gamma))$, and the CTL model checking problem is in **P** [12]. □

5. BOUNDED LOOKAHEAD

The application of guards assumes that in addition to a history, the guard has access to the state transition system. Using the information in the transition system, we attempt to construct guard functions that given a history, rule out certain successors of the last state on the history (intuitively, those leading to a norm violation). In this section, we consider a setting with *bounded lookahead*. Bounding the lookahead reduces the amount of computation required to enforce the norm, by restricting the ability of the normative organisation to reason about and to take into consideration the future evolution of the system. In Section 6 we consider the problem of the minimal size of the lookahead window required to perfectly enforce a given norm on a given model. In this section, we introduce the formal machinery to define guards which operate with a bounded lookahead, in contrast to guards which have unbounded lookahead. We introduce the notion of a *view*, as the part of a run considered by a guard. The length of a view is defined by a *window*. To simplify the presentation, we assume that windows only restrict the lookahead, and that all past events are remembered and can be perfectly observed. Note that a past-restricting window could be seen as a way to limit the amount of memory the normative organisation can use to store information about the past. While restricting the past window is important for resource-bounded systems, we leave it for future work. In the remainder of this section we again assume that $M = (S, R, V)$ is a finite transition system.

5.1 Windows and Views

We consider the case of a contiguous view consisting of the sequence of states before and some finite sequence of states after the current position on a run ρ. A *window* defines the size of the view and is just a natural number defining the future lookahead. A *view* maps a run together with the current position on the run to a history. Given a run ρ and a position i, the view $\mathsf{view}(\rho, i)$ with respect to window W returns the history consisting of all past events $\rho[0, i-1]$, the current state $\rho[i]$ and the future states $\rho[i, i+W]$ which fit into the window.

DEFINITION 13 (WINDOW AND VIEW). *A* (future) window *is given by $W \in \mathbb{N}$. Let ρ be a run, W a window and $i \in \mathbb{N}_0$ (the current position on ρ). The* view of ρ at i with respect to W, $\mathsf{view}^W(\rho, i)$, *is defined by the function:* $\mathsf{view}^W : \mathcal{R}_M \times \mathbb{N}_0 \to \mathcal{H}_M$ *with* $(\rho, i) \mapsto \rho[0, i+W]$. *We also lift the view with respect to a window to a set of runs X:* $\mathsf{view}^{\mathsf{W}}(X, i) = \bigcup_{\rho \in X} \mathsf{view}^{\mathsf{W}}(\rho, i)$

Consider for example the run $\rho = s_1 s_2 s_3 s_4 s_5 s_6^{\omega}$. Then, we have that: $\mathsf{view}^2(\rho, 1) = s_1 s_2 s_3 s_4$. The part $s_1 s_2$ is the current history where s_2 is the current state; and $s_3 s_4$ encodes the lookahead. The window has size 2.

5.2 Window-Filtered Guards

We now integrate windows with guards and their application. A guard applied in context of a window can only be evaluated on the current view and thus has to decide which actions to disable based on a history rather then on a run. The *W-filtered guard* of a guard formula γ takes as an input a history h and returns a set of enabled states. In order to give the formal definition we first need to introduce a *finite trace semantics* for LTL^{+p}-formulae.

DEFINITION 14 (FINITE TRACE SEMANTICS). *We define a finite trace semantics for LTL^{+p} with two truth values (t and f). Given a transition system M, a history h, and a position i $0 \leq i < h$, the truth of LTL^{+p} formulas is defined by the semantic relation $\models^f$:*

$M, h, i \models^f p$ *iff* $p \in V(\rho[i])$

$M, h, i \models^f \neg\phi$ *iff* $M, h, i \not\models^f \phi$

$M, h, i \models^f \phi \wedge \psi$ *iff* $M, h, i \models^f \phi$ *and* $M, \rho, i \models^f \phi$

$M, h, i \models^f Y\phi$ *iff* $i > 0$ *and* $M, \rho, i-1 \models^f \phi$,

$M, h, i \models^f \phi S \psi$ *iff* $\exists j, 0 \leq j \leq i$ *s.t.* $M, h, j \models^f \psi$ *and* $\forall k : j < k \leq i : M, h, k \models^f \phi$

$M, h, i \models^f X\phi$ *iff* $i < |h|$ *and* $M, \rho, i+1 \models^f \phi$

$M, h, i \models^f \phi U \psi$ *iff* $\exists j, i \leq j < |h|$ *s.t.* $M, \rho, j \models^f \psi$ *and* $\forall k : i \leq k < j : M, \rho, k \models^f \phi$

This semantics captures an essentially pessimistic view, meaning that *false positives* are not acceptable. This semantics is used to interpret guards on a finite trace because a transition should only be enabled if the guard is *definitely true* on the view. We note that if used for (future-time) LTL-formulae, this semantics coincides with the finite trace semantics given in [14].

We also note that the finite trace semantics would not always be appropriate when evaluating norms on a finite trace in order to detect norm violations. In this case, a violation should only be reported if the violation has or will definitely take place, that is *false negatives* are not appropriate. A formal definition of such a semantics can be easily obtained by slightly modifying the truth conditions given for $\models^f$. In the following we shall denote the 'optimistic variant' of $\models^f$ by $\models^f_o$.

We can now formally define the notion of window-filtered guard. Given a guard formula γ the *W-filtered guard* $G_M^{\gamma,W}$ returns, for an input history h, all states s such that there is a run ρ which extends hs in such a way that the history $\mathsf{view}^{\mathsf{W}}(\rho, |h|-1)$ satisfies the guard γ. For convenience, the evaluation of the guard takes place in the last state of the view. We note, however, that the guard formula γ could equivalently be evaluated at position $|h|$ on $\mathsf{view}^{\mathsf{W}}(\rho, |h|-1)$ as done in the unbounded setting.

DEFINITION 15 (WINDOW-FILTERED GUARDS). *Let γ be a guard formula and W a window. The W-filtered guard $G_M^{\gamma,W}$ is defined as follows:*

$$G_M^{\gamma,W}(h) = \{s \in S \mid \exists \rho \in \mathcal{R}_M : hs \sqsubset \rho \text{ and } M, \mathsf{view}^{\mathsf{W}}(\rho, |h|-1), |h|+W-1 \models^f \gamma\}$$

for all histories $h \in \mathcal{H}_M$. Analogously, $G_M^{\gamma,W}$ is called deadlock-free *if it is non-empty for all consistent histories h (now with respect to the W-filtered guard). The* application *of a W-filtered guard is defined as before, where we write $\mathcal{R}_M^{\gamma,W}$ instead of $\mathcal{R}_M^{G_M^{\gamma,W}}$.*

EXAMPLE 5 (EXAMPLE 2 CONT.). *We consider the W-filtered guard $G' = G_{M_1}^{\gamma_3,W}$, with $\gamma_3 = Gp$. For $W = 3$ the guard G' (perfectly) enforces $(\varphi_1)_{M_1}$. Let us consider $G'(s_0)$. The views of the two possible runs from s_0 are $s_0 s_1 s_2 s_3$ and $s_0 s_4 s_5 s_5$. The first view is discarded as $M_1, s_0 s_1 s_2 s_3, 3 \not\models^f \gamma_3$. It is also easy to see that G' is deadlock-free for $W = 3$. For all $W < 3$ however, the guard G' is not deadlock free. Suppose e.g. $W = 2$. Then $s_1 \in G'(s_0)$ as $M_1, s_0 s_1 s_2, 2 \models^f \gamma_3$. In the next step, however, we have $G'(s_0 s_1) = \emptyset$ as $M_1, s_0 s_1 s_2 s_3, 3 \models^f \gamma_3$, and the system deadlocks after the history $s_0 s_1$.*

A key problem is to determine (a minimal) window W and a (as general as possible) guard γ such that $G_M^{\gamma,W}$ does (optimally) enforce a given norm φ_M. We consider this problem in the next section.

6. PRACTICAL ENFORCEMENT

In this section we study some practical aspects of norm enforcement. First, we show that, in general, norm enforcement is a difficult problem. Then we study canonical guards for bounded lookahead for liveness and safety properties. We show that the canonical guard can always enforce such norms if the window size is sufficiently large. We also look at other, more practical, guard functions with bounded lookahead.

6.1 The General Problem

The first problem we investigate is whether a guard function enforces a norm given unbounded lookahead. We show that this problem is intractable.

PROPOSITION 7 (ENFORCEMENT: UNBOUNDED CASE). *Let M be a transition system, G^{γ}_{M} be a LTL^{+p}-defined df-guard, and φ an LTL-defined norm. The problem whether G^{γ}_{M} enforces φ_M is **PSPACE**-hard and in **2EXPTIME**.*

PROOF IDEA. The guard γ is a pure LTL^{+p}-formula and φ is an LTL-formula; thus, $A(GXE\gamma \rightarrow \varphi)$ is a $\mathsf{CTL}^* + \mathsf{past}$-formula. The model checking problem for $\mathsf{CTL}^* + \mathsf{past}$ is shown to be in **2EXPTIME** [10]. Moreover, $M, s \models A(GXE\gamma \rightarrow \varphi)$ for all $s \in S_0$ iff G^{γ}_{M} enforces φ_M. To see this we analyse the formula: $M, s \models A(GXE\gamma \rightarrow \varphi)$ iff $\forall \rho \in \mathcal{R}_M(s)[(\forall i \forall \exists \rho' \in \mathcal{R}_M, \rho[0,i] \sqsubset \rho', M, \rho', i \models \gamma) \Rightarrow M, \rho, 0 \models \varphi]$ iff $\forall \rho \in \mathcal{R}_M(s)[(\forall h s' \in \mathcal{H}_M, hs' \sqsubset \rho, s' \in G^{\gamma}_{M}(h)) \Rightarrow M, \rho, 0 \models \varphi]$ iff $\forall \rho \in \mathcal{R}_M(s)[\rho \in \mathcal{R}^{\gamma}_{M}(s) \Rightarrow \rho \in \varphi_M]$.

Hardness. For $\varphi = \bot$ and $\gamma = P(Y\bot \wedge \gamma')$ for some LTL^{+p}-formula γ' we have: $M, s \models A(GXE\gamma \rightarrow \varphi)$ iff $M, s \models A(GXE\gamma \rightarrow \bot)$ iff forall runs $\rho \in \mathcal{R}_M(s)$, $M, \rho, 0 \not\models \gamma'$ iff there is no run $\rho \in \mathcal{R}_M$ with $M, \rho, 0, \models \gamma'$. The latter problem corresponds to (universal) model checking the LTL^{+p}-formula γ'. A **PSPACE**-complete problem [23]. □

In the case of bounded lookahead, a norm can be enforced given a window of size k, if it is possible to determine by looking only k steps ahead, whether the norm must be violated on all paths allowed by the guard from a given state s. Conversely, a norm cannot be enforced given a window of size k and a guard G if all paths consistent with the guard G from a state violate the norm but only after at least k steps. In that case, a transition is made which will inevitably violate the norm at some later moment.

Suppose the current history is h and we need to decide whether a transition to s should be allowed. For any LTL norm φ, this problem can be solved in a straightforward (but computationally expensive) way by checking: (1) for each finite trace t in the k-unraveling from the current state s, whether $M, ht, 0 \models^{f}_{o} \varphi$ (using the 'optimistic' finite trace semantics; that is, a violation is only reported if there is no way to satisfy the norm); (2) $M, ht, |h| + |t| - 1 \models^{f} \gamma$ and (3) $M, \rho, 0 \not\models \varphi$ for all ρ with $hs \sqsubset \rho$. That is, whether a transition to s inevitably results in the violation of the norm, but this cannot be detected in k steps. Such an approach has exponential complexity.

These results show that norm enforcement is in general a difficult problem. In the remainder of this section, we consider the special cases of *safety* and *liveness norms*. We use $LTL^{+}(T_1, T_2, \ldots)$ to refer to the fragment of LTL which only allows the temporal operators $T_1, T_2, \ldots$, and only outside the scope of negation. These fragments can be used to define safety and liveness properties:

DEFINITION 16 (SAFETY AND LIVENESS NORM). *We say that an LTL-formula φ is a* safety norm *if $\varphi \in LTL^{+}(G, X)$. The formula is a (finite)* liveness norm *if $\varphi \in LTL^{+}(F, U)$.*

For example, the formula $G(p \rightarrow Xq)$ is a safety property, and Fp and pUq are simple liveness properties.

We define prefixes which can be used to witness the truth/or non-truth of such a property with respect to a given model M. For a (finite) liveness property φ we say that $h \in \mathcal{H}_M$ is a *good prefix of φ in M* if for all $\rho \in \mathcal{R}_M$ with $h \sqsubset \rho$, $M, \rho, 0 \models \varphi$. Similarly, for a safety property φ we say that $h \in \mathcal{H}_M$ is a *bad prefix of φ in M* if for all $\rho \in \mathcal{R}_M$ with $h \sqsubset \rho$, $M, \rho, 0 \not\models \varphi$. We note that for a liveness property φ, each $\rho \in \varphi_M$ has a good prefix; and for a safety property φ, each $\rho \notin \varphi_M$ has a bad prefix (cf. [8] for a general discussion).

Unfortunately, even for safety and liveness properties the problem presented in Proposition 7 remains intractable. Even if the guards are given by Boolean formulae the problem is **PSPACE**-complete. This follows from [21], where it is shown that model checking formulae of $LTL^{+}(G, X)$ as well as of $LTL^{+}(U)$ is **PSPACE**-complete.

Given this observation, an interesting question is, whether we can construct a guard from which we *know* that it will enforce a safety property, without having to check whether it actually enforces the norm. More precisely, we consider two types of results:

- We show that we can compute an upper bound such that the canonical guard always enforces the norm. At *run-time* we simply have to check whether the canonical guard is satisfied. It is important to note, that this does not require *model-checking* the guard, which would again be intractable. Other techniques, like progression or automata theoretic approaches can be used to efficiently monitor the truth of safety and liveness norms [9]. Propositions 8, 9, and 12 are in line with this view.
- We show how to *precompute* a guard function which operates by simply disabling successor states which are in a list of prohibited states. Proposition 10 is a result in this direction.

6.2 Enforcement of Safety Norms

The next proposition gives a positive answer to the question above: if we choose a window of sufficient size, then the canonical guard for a safety norm guarantees that the norm will be enforced, moreover the enforcement is *perfect*.

Let φ be a safety norm of type $\varphi = Gp$. Then, a state s should be disabled from the current history h, if Gp cannot be ensured on any path from s. In order for a path satisfying Gp to exist, we need to find a cycle on which Gp holds. If such a cycle exists, it must be found within $|S|$ steps. This is an upper bound for the lookahead needed to decide whether safety norm $\varphi = Gp$ will be violated if state s is reached. For arbitrary safety norms, the reasoning for computing the lookahead is similar.

PROPOSITION 8 (SAFETY NORM: PERFECT ENFORCEMENT). *Let φ be an LTL-safety property and M a transition system. One can compute a window W such that the canonical guard of φ enforces perfectly φ_M under window size W.*

Earlier in Example 5 we have seen that, even with very simple norms such as Gp, enforcement may require arbitrary long lookahead windows, depending on the size of the model. However, depending on the class of models the lookahead can be significantly reduced as illustrated by the next example.

EXAMPLE 6 (CLASS OF MODELS WITH LOOKAHEAD ONE). *There is a special class of model where Gp can be perfectly enforced with a lookahead window of just one. This holds for models M that are reflexive and have at least one run from each state in*

S_0 satisfying Gp. For such an M, a df-guard for Gp can be defined as follows: $G_M(h) = \{s \mid (h[\infty], s) \in R \text{ and } p \in V(s)\}$ Clearly, this guarantees that the next state always exists (since $(h[\infty], h[\infty]) \in R$) and all states on runs consistent with G_M satisfy p.

We now show how to determine the minimal size of a lookahead for a simple kind of safety norms.

DEFINITION 17 (STATE-BASED SAFETY NORM). *A state-based safety norm is of the form $G\psi$ where $\psi \in LTL^+(X)$.*

State-based safety norms have a useful property: it is possible to define a set of states from which all paths inevitably violate the norm (do not satisfy ψ). Note that this is a CTL-definable property, using only AX operators. We call this set of states $Viol_M(\varphi)$.

EXAMPLE 7. *An example of a state-based safety norm is Gp. For this norm, the set $Viol_M(\varphi)$ is the set of states where p is false.*

Similarly, for a state-based safety norm $\varphi = G(p \rightarrow Xq)$ the set $Viol_M(\varphi)$ consist of all states s such that $p \in V(s)$ and for all states s' reachable from s in one step, $q \notin V(s')$. This is definable as $p \wedge AX\neg q$.

Clearly, a guard function for φ should not return successors in $Viol_M(\varphi)$. It should also disable states which are not themselves in $Viol_M(\varphi)$, but from which all paths lead inevitably to a $Viol_M(\varphi)$ state. We call the latter set of states $Viol^+_M(\varphi)$ states. This latter set is also CTL-definable as $AF\chi$ where χ describes $Viol_M(\varphi)$. To enforce a state-based safety norm, we need to avoid states in $Viol^+_M(\varphi)$.

In the following, we show how to compute the minimal window size k of Proposition 8, such that the canonical guard of a state-based safety norm enforces the norm. We show that the complexity of finding the minimal k is polynomial.

We give an algorithm to compute the minimum window size for enforcing φ on M (see Algorithm 1). The algorithm works by computing the set of states $\tau = Viol^+_M(\varphi)$ from which all paths lead to $Viol_M(\varphi)$. Starting from the states in $Viol_M(\varphi)$, υ, we repeatedly compute the set of transitions on a path leading to υ states, δ, the states τ from which all transitions lead to a υ state or a τ state, and the set of simple paths to states in τ, σ. Finally we compute the longest simple path between two states in τ, which gives us the required lookahead k.

Note that $Viol_M(\varphi)$ can be computed using CTL model-checking in time polynomial in the size of the φ and M. It is also possible to compute τ by computing the set of states satisfying $AF\chi$ where χ describes $Viol_M(\varphi)$. Instead we give an explicit algorithm, since it also computes the size of the longest simple path from a state to a state in τ. The complexity of Algorithm 1 is $O(|S| \times |R|)$.

From the algorithm the following proposition immediately follows:

PROPOSITION 9. *The problem of deciding whether window size k is sufficient for the canonical guard to enforce a state-based safety norm on a model M is in **P**.*

The most straightforward way to solve this problem is to find the minimal size of the window and compare it to k. We can also define a decision variant of Algorithm 1, which simply checks whether a given window size is sufficient to enforce the norm. Rather than looping until τ does not change, we repeat the loop $k+1$ times (for a window size of k), and check if the length of the longest path returned is $> k$. However given the complexity of the algorithm, it is not clear that the reduction in computation is worthwhile.

Algorithm 1 Computing window size for state-based norm φ

function WINDOW-SIZE($\varphi, M = \langle S, R, V\rangle$)
  $\upsilon \leftarrow Viol_M(\varphi)$
  $\delta \leftarrow \{(s,(s,s')) \mid (s,s') \in R \wedge s' \in \upsilon\}$
  $\tau \leftarrow \{s \mid \forall (s,s') \in R\ ((s,(s,s')) \in \delta)\}$
  $\sigma \leftarrow \{(s, s \circ [s']) \mid s \in \tau \wedge (s,(s,s')) \in \delta\}$
  $\epsilon \leftarrow \{(s,(s,s')) \mid (s,s') \in R \wedge s' \in \upsilon \cup \tau\}$
  while $\delta \subset \epsilon$ **do**
    $\delta \leftarrow \epsilon$
    $\tau \leftarrow \tau \cup \{s \mid \forall (s,s') \in R\ ((s,(s,s')) \in \delta)\}$
    $\sigma \leftarrow \{(s, s \circ \rho) \mid s \in \tau \wedge \rho \in \underset{(s,(s,s')) \in \delta,\ (s',\rho) \in \sigma,\ s \notin \rho}{\arg\max} |\rho|\}$
    $\epsilon \leftarrow \{(s,(s,s')) \mid (s,s') \in R \wedge s' \in \upsilon \cup \tau\}$
  end while
  return $\arg\max_{(s,\rho)\in\sigma} |\rho|$
end function

An alternative to computing the size of the window and using it at run-time in combination with the canonical guard, is to use the set of 'bad' states $Viol^+_M(\varphi)$ directly in implementing the enforcement of the norm. The guard is defined as follows:

$$G_M(h) = \{s' \mid (h[\infty], s') \in R \text{ and } s' \notin Viol^+_M(\varphi)\}$$

where $Viol^+_M(\varphi)$ is τ computed by Algorithm 1. We note that the guard can be implemented by looking only at the last state of the view, that is, whether the last state is prohibited (is in $Viol^+_M(\varphi)$). In this approach, enforcement effectively requires a window of size 1, since all that is required is to check if the state resulting from a transition is in $Viol^+_M(\varphi)$, so we have:

PROPOSITION 10. *State-based safety norms can be perfectly enforced by a guard which only requires one step lookahead.*

6.3 Enforcement of Liveness Norms

In Proposition 8 we have shown that safety norms can be perfectly enforced. This is not the case for liveness norm:

PROPOSITION 11. *Let M be a transition system. In general, an LTL-liveness property φ cannot be perfectly enforced.*

PROOF. In Example 3 we have shown that the norm $\varphi_0 = Fp$ which is a liveness property in M_0 is not enforced by the canonical guard $\gamma_{\varphi_0} = P(Y\bot \wedge \varphi_0)$. Thus, by Proposition 5 φ cannot be perfectly enforced by any guard. □

However, we can compute a lookahead that allows us to enforce (though not perfectly or optimally), a liveness norm. Let φ be a liveness property. Then, for each initial state $s \in S_0$, there is a minimal length good prefix of φ in M. We define function $d_M : S_0 \rightarrow \mathbb{N}^\infty_0$ which assigns to each initial state the length of a minimal length good prefix of φ. If such a history does not exist we set $d_M(s) = \infty$. That is, from each state s_0 the norm can be satisfied in $d_M(s_0)$-many transitions. Now, we define $L^\varphi_M = \max\{d_M(s) \mid s \in S_0, d_M(s) \neq \infty\}$ if for some $s \in S_0$ we have that $d(s) < \infty$; and $L^\varphi_M = \infty$ otherwise. Consequently, a lookahead of L^φ_M would be sufficient do decide whether a transition to the a next state should be enabled or not, if it is possible to satisfy the norm at all. So, the good news is that, in general, simple liveness norms can be enforced. Moreover, an upper bound for L^φ_M is $|\varphi| \cdot |S|$. This can be seen by decomposing φ into at most $|\varphi|$ independent checks of simpler liveness formulae. Each of these formula can be witnesses within $|S|$ steps in the model.

PROPOSITION 12 (LIVENESS CAN BE ENFORCED). *Let φ be an LTL liveness property and M a transition system. The canonical-guard of φ enforces φ_M under window size $W = L_M^\varphi$; in general, the enforcement is neither optimal nor perfect. An upper bound for L_M^φ is $|\varphi| \cdot |S|$.*

In order to use this result, we can either use the upper bound or compute the minimal lookahead L_M^φ. For liveness norms of type $\varphi = Fp$ and $\varphi = q\,Up$, for example, the value L_M^φ can be computed in polynomial time in the size of the model. For both types of formulae the upper bound for L_M^φ is $|S|$. Moreover, in both cases, we can use Dijkstra's algorithm [15] to compute the shortest path between an initial state s and a state s' with $p \in V(s')$. In the case of $\varphi = qUp$ the search simply ignores states s' with $\{q, p\} \cap V(s') = \emptyset$.

7. RELATED WORK

In the literature on multi-agent systems, many computational and programming frameworks for norms and norm enforcement have been proposed. Existing frameworks mainly focus on specific types of norms, and norm monitoring and enforcement mechanisms, e.g., [18, 16, 6, 13]. Of these, our approach is closest to that of [16]; however their approach does not support prediction of future violations, or attempt to ensure that violations do not occur. The focus of our paper is different, as we are interested in a formally defined computational model for a run-time enforcement mechanism that monitors LTL norms to prevent future norm violations at run-time. In particular, we are interested in formal models of run-time norm enforcement and their computational complexity. Our model can be applied to develop computational frameworks for run-time norm enforcement mechanisms.

Our work is closely related to [1] where multi-agent systems are modelled as transition systems and norm regimentation is modelled by removing bad transitions. However, in contrast to our approach, [1] considers norms semantically by means of a set of bad transitions that are removed at design time by updating the transition system with norms. In our work, norms are specified syntactically by means of LTL formulas, and bad transitions are disabled at run-time by means of a guard function.

Another related work on norm monitoring is [11, 5], where norms are considered in the context of imperfect monitors. Similar to our work, [11, 5] considers norms syntactically by means of LTL formulas, and monitors are assumed to be imperfect. However, [11, 5] consider imperfect monitors at an abstract level and by means of an indistinguishability relation between runs, while we consider imperfect monitors at a practical level and by means of a windowing mechanism. In particular, the window mechanism in our approach restricts the computational effort required to predict the future of the actual run. Moreover, [11] ignores the problem of run-time norm monitoring and focus on the notion of imperfect monitors, while [5] focuses on the problem of synthesising the closest approximation of a norm that can be monitored with a set of imperfect monitors. Finally, imperfect monitors in [11] can be improved by means of their combinations, while in our approach an imperfect monitor can be improved by increasing the size of its window.

It is obvious that our work is closely related to, and actually inspired by, the run-time verification literature, e.g., [9]. We build on this work to evaluate norms (LTL properties) on finite runs, but in contrast to run-time verification literature, we consider the construction of a guard function that enforces norms by avoiding norm violations. Our work also extends run-time verification approaches by proposing lookahead and windowing mechanisms to predict violations of LTL properties. Our work is also closely related to supervisory control theory for discrete event systems, e.g., [22, 7]. While supervisory control theory approaches the problem of synthesising control mechanisms at an abstract level by means of operations on formal languages (e.g., intersection, inclusion, product), our approach is more practical focusing on the construction of guard functions that disable violating transitions at run-time.

We would like to emphasise that our work differs from work on the verification of normative updates and and the compliance of protocols with norms as proposed in [17, 4, 13, 2]. One difference is that we focus on run-time norm enforcement while [17, 4, 13, 2] consider offline verification of normative updates and protocols. Second, [4] consider specific norms (conditional prohibitions/obligations with deadline) that can be expressed in a fragment of CTL* with Past, while we consider various types of norms that can be expressed by various fragments of LTL. Third, [4, 13] consider norm enforcement at design time by means of both regimentation and sanctioning, while our work focuses on run-time norm regimentation. Enforcing norms by sanctions allows norm to be violated, but imposes sanctions when norms are violated. We believe that our approach can be extended to model enforcement by sanctions. This can be done by, for example, imposing sanctions on the states that are reached by violating transitions, instead of disabling those transitions. We consider norm enforcement by means of sanctions as a future direction of our research.

In this work, we have focused on norm regimentation from an organisational perspective. Accordingly, the guard function is considered as a functionality of the agents' organisation, which enables/disables the options that are available to the agents at run-time. We did not focus on norm regimentation mechanism from an agent perspective and have ignored issues such as agents' norm awareness as studied in [24, 3]. Of course, our work can be applied to make individual agents norm aware in the sense that the agents themselves compute the options that cause norm violations. Using this ability the agents can decide whether to violate or to obey the norms at run-time. We consider the norm awareness issue as a future direction of our research.

8. CONCLUSIONS

This paper presents a formally defined computational model for run-time norm enforcement mechanism. The model is based on a guard function that enables/disables options that (could) violate norms after a system history. The guard function, which is characterised by LTL formulae with past operators, may use computational resources to reason about the future of the actual run in order to disable the options that cause norm violations. The computational resources that allow to reason about the future of the run is modelled by means of a window of a particular size. The model is formally analysed and the computational complexity for various types of run-time norm enforcement is provided.

Our approach can be extended in various directions. A possible extension would be to allow norm enforcement by means of sanctions. In this regard, the interaction between the predication of norm violations and sanctioning becomes an interesting but challenging issue. Another possible extension is to allow windows with limited size in the past. In this paper, a window covers the complete history from a starting state, but this is obviously not a practical assumption as this requires sufficient memory capacity to store the history; so we have already investigated cases where the full history is not used. Limiting the history size of a window may cause some norm violations to be undetected and not enforceable. Finally, our model can be used to allow agents to evaluate and to reason about the system behaviours in order to decide whether they obey or violate norms.

REFERENCES

[1] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Robust normative systems and a logic of norm compliance. *Logic Journal of the IGPL*, 18(1):4–30, 2010.

[2] H. Aldewereld, J. Vázquez-Salceda, F. Dignum, and J. C. Meyer. Verifying norm compliancy of protocols. In O. Boissier, J. A. Padget, V. Dignum, G. Lindemann, E. T. Matson, S. Ossowski, J. S. Sichman, and J. Vázquez-Salceda, editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2006.

[3] N. Alechina, M. Dastani, and B. Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1057–1064. IFAAMAS, 2012.

[4] N. Alechina, M. Dastani, and B. Logan. Reasoning about normative update. In *Proceedings of the Twenty Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 20–26. AAAI Press, 2013.

[5] N. Alechina, M. Dastani, and B. Logan. Norm approximation for imperfect monitors. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, 2014.

[6] S. Alvarez-Napagao, H. Aldewereld, J. VÃązquez-Salceda, and F. Dignum. Normative monitoring: Semantics and implementation. In M. De Vos, N. Fornara, J. Pitt, and G. Vouros, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, volume 6541 of *Lecture Notes in Computer Science*, pages 321–336. Springer Berlin Heidelberg, 2011.

[7] G. Aucher. Supervisory control theory in epistemic temporal logic. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, 2014.

[8] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, May 2008.

[9] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14:1–14:68, 2011.

[10] L. Bozzelli. The complexity of CTL*+ linear past. In *Foundations of Software Science and Computational Structures*, pages 186–200. Springer, 2008.

[11] N. Bulling, M. Dastani, and M. Knobbout. Monitoring norm violations in multi-agent systems. In *Twelfth International conference on Autonomous Agents and Multi-Agent Systems (AAMAS'13)*, pages 491–498, 2013.

[12] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[13] M. Dastani, J.-J. C. Meyer, and D. Grossi. A logic for normative multi-agent programs. *Journal of Logic and Computation*, 23(2):335–354, 2013.

[14] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. AAAI Press, 2013.

[15] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[16] M. Esteva, J. Rodríguez-Aguilar, B. Rosell, and J. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of AAMAS 2004*, pages 236–243, New York, US, July 2004.

[17] M. Esteva, J. Rodriguez-Aguilar, C. Sierra, and W. Vasconcelos. Verifying norm consistency in electronic institions. In V. Dignum, D. Corkill, C. Jonker, and F. Dignum, editors, *Proceedings of the AAAI-04 Workshop on Agent Organizations: Theory And Practice*, volume Technical Report WS-04-02, San Jose, July 2004. AAAI, AAAI Press.

[18] J. Hübner, J. S. Sichman, and O. Boissier. $\mathcal{S}$-$\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In *Proceedings of the international workshop on Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer, 2006.

[19] F. Laroussinie and P. Schnoebelen. Specification in CTL+ past for verification in CTL. *Information and Computation*, 156(1):236–263, 2000.

[20] N. Markey. Temporal logic with past is exponentially more succinct. *EATCS Bulletin*, 79:122–128, 2003.

[21] N. Markey. Past is for free: on the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6-7):431–458, 2004.

[22] P. Ramadge and M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77(1), 1989.

[23] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[24] M. B. van Riemsdijk, K. V. Hindriks, and C. M. Jonker. Programming organization-aware agents: A research agenda. In *Proceedings of the Tenth International Workshop on Engineering Societies in the Agents' World (ESAW'09)*, volume 5881 of *LNAI*, pages 98–112. Springer, 2009.