

Designing Protocols for Agent Institutions*

Huib Aldewereld[†]
Institute of Information and
Computing Sciences
Utrecht University
The Netherlands
huib@cs.uu.nl

Frank Dignum
Institute of Information and
Computing Sciences
Utrecht University
The Netherlands
dignum@cs.uu.nl

John-Jules Ch. Meyer
Institute of Information and
Computing Sciences
Utrecht University
The Netherlands
jj@cs.uu.nl

ABSTRACT

We show how protocols can be derived from norms using landmarks. The resulting protocols can be used by agents to fulfill the norms governing an e-institution without having to have a capability for normative reasoning. It can also be used by normative agents as a default protocol to be used, but from which it can deviate in specific circumstances.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*multiagent systems*

General Terms

Theory, Legal Aspects, Design

Keywords

Norms, Electronic Institutions, Normative Systems

1. INTRODUCTION

Agent-mediated institutions (or e-institutions), introduced in [?, ?], are *open* agent systems that allow heterogeneous agents to enter and perform tasks. The e-institutions specify the admissible behavior of the agents by means of norms, which are declarative and abstract by nature. On the one hand this allows for a stable specification suitable for almost any conceivable situation that arises in the institution, but in the other hand the norms hardly give any indication which interaction patterns would guarantee satisfaction of the norms.

One could aim for the use of normative agents in e-institutions as propagated in [?]. In this perspective the agents are capable of reasoning about the norms and planning their actions

*This is a student paper.

[†]The first author of this paper was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 634.000.017

accordingly. However, it seems not very realistic that all agents will have this capacity. Most agents will be standard agents that are only capable to reason about standard protocols as part of their interactions with other agents. Therefore, in this paper, we aim to provide ways to generate protocols (on the basis of the normative descriptions) that are guaranteed to fulfill all norms of the e-institution. Given these protocols agents entering the e-institution can just follow these protocols and be sure they will always stay within the "law". Note that we do not necessarily require the agents to follow these protocols. They can be seen as available templates for use by the agents. Agents can still perform normative reasoning if they are capable, but, with the help of these protocols, they do not need to do that in order to participate in the e-institution.

Our approach is inspired by how the gap between the normative and procedural dimensions is bridged in human institutions. Human laws express in a very abstract way wanted (legal) and unwanted (illegal) states of affairs. Although laws are very expressive, they do not express how to achieve a given state of affairs, and therefore they are very hard to use in practise to, e.g., guide each decision in a process. In practise more efficient representations are needed, such as protocols or guidelines. In rule-based legal systems (those based in Roman-Germanic law), *regulations* add an intermediate level between laws and practise, by giving some high-level specifications on some constraints about how things can or cannot be done. These high-level descriptions are therefore interpretations of the law that add some operational constraints to be met by the practise (see figure 1). Using this idea, we introduce an intermediate level between institutional norm specifications and institutional protocols based on *landmarks* in a similar way as done in [?, ?]. The landmarks are further discussed in section 2.

From the norms we will automatically generate finite state automata, using a technique introduced by Wolper in [?]. This is a general technique to convert temporal logic formulas in LTL into so-called Büchi automata. The landmark patterns can be obtained from these automata by looking at their corresponding recognized languages. This technique is described in section 3. In section 4 we show how we can compose protocols by strengthening the landmark patterns, through adding additional information concerning the (presumed) capabilities of the agents. In section 5 an example of the whole process of generating a protocol from norms is given. The paper concludes with some observations and future in research in section 6.

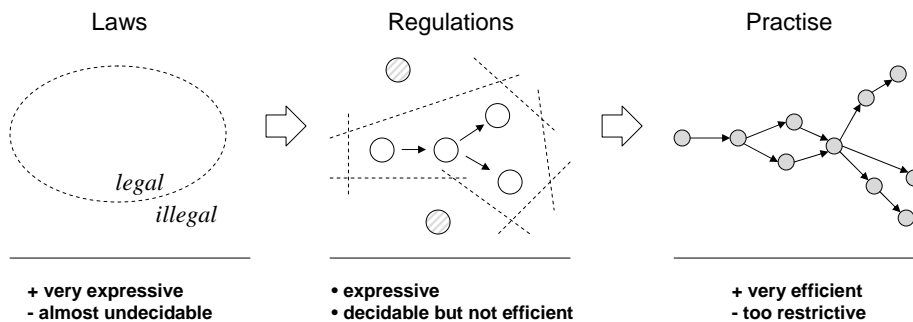


Figure 1: Comparison between Laws, Regulations and Practise

2. NORMS AND LANDMARKS

No matter how landmarks are represented – as states, or sets of states – their relevance in protocol specification is dictated by the simple observation that several different actions can bring about the same outcome. Once the outcomes of actions are organised in a structured description (i.e. a landmark pattern), it becomes possible to represent families of protocols abstracting from the actual transitions by which each protocol is constituted. This makes landmarks an ideal solution for bridging the gap between the abstract normative level and the procedural level of protocols. The landmark pattern fully captures the order in which states should occur, representing the important steps that any protocol should contain, while still abstracting from the actual procedural information on how the transition from one state to another should be achieved. In essence, a landmark pattern represents “those steps that should be taken and in which order”.

Formally:

DEFINITION 1 (LANDMARK PATTERN).

A landmark pattern is a structure $\mathcal{L} = L^+, \leq$ where L^+ is a finite set of landmarks and \leq is a partial order on L^+ .

Similar to the relation between norms, regulations and practise, where regulations add operational information to the restrictions given by the norms, a landmark pattern should add additional information to the normative goals in order to bridge the gap between the norms and the practise. The norms only give a (temporal) ordering of the states that should be reached. The normative landmark pattern extracted from these norms will thus leave many blanks, situations where the order of events/actions is undetermined by the norms or only minimally described. The choices in ordering that are not specified in the pattern, however, will in most cases influence the efficiency or even the feasibility of the pattern. For example, norms concerning organ transplantation describe that permission for organ removal must be obtained before the organs are removed, as well as that doctors should check whether a patient is brain death before starting the operation for removing organs of the patient. These two norms only describe that two states should be reached (*permission is obtained*, ρ , and *patient’s death is certified*, δ) before another state happens (*organs are removed*, π). No information is given about the ordering of states ρ and δ , but obviously making sure that δ happens before ρ is more practical than having ρ occur before δ . This would mean that we add the ordering $\delta \leq \rho$ to our landmark pattern, to obtain $\langle \{\rho, \delta, \pi\}, \{\rho \leq \pi, \delta \leq \pi, \delta \leq \rho\} \rangle$.

In this case we added an extra ordering between existing landmarks to exclude the possibility of inefficient situations arising, but additional landmarks, both positive as well as negative ones, can be added to the existing pattern to express information concerning efficiency and feasibility. Take for instance an extension of our previous example, where we include the additional step of assigning the organs that were just extracted from the deceased patient. According to the law, the assignment procedure can only start after the organs have become available (though, in this case, it is in dispute whether an organ is available after the permission has been given to extract the organ, or after the actual extraction, for simplicity we assume the latter is the case), giving a landmark pattern (α represents the state of the organ being assigned): $\langle \{\rho, \delta, \pi, \alpha\}, \{\rho \leq \pi, \delta \leq \pi, \delta \leq \rho, \pi \leq \alpha\} \rangle$ ¹. The procedure of assigning organs, however, needs certain information about the organ to make sure that the assigned receiver is a compatible recipient. Leaving the pattern as it currently is, this information will need to be gathered at the moment the assignment is taking place (meaning, somewhere after π , but before or at the time of occurrence of α). Most of the information needed for the assignment process (for instance, the blood type of the deceased, which influences the list of compatible recipients) can be gathered in the very beginning of the protocol (after or at the moment one checks whether the deceased is medically death), and doing so can greatly improve the speed at which a organ can be assigned, because a compatible organ recipient is found earlier in the process and thus increases the chances of the success of the organ transfer (due to degradation of the organ once it is recovered from the deceased patient). This means that adding a landmark (*check bloodtype*, β) to the beginning of the landmark pattern can increase the efficiency of the existing pattern. The resulting pattern would, for example, be $\langle \{\rho, \delta, \pi, \alpha, \beta\}, \{\rho \leq \pi, \delta \leq \pi, \delta \leq \rho, \pi \leq \alpha, \delta \leq \beta, \beta \leq \rho\} \rangle$.

3. FROM NORMS TO LANDMARKS

Creating a protocol for a normative domain is done through the use of the intermediate level of landmarks that we presented above. This process of generating a (proto-typical) protocol for a normative domain is the following. First a set of landmarks is extracted from the norms governing the domain. To extract this (normative) landmark pattern from the norms we use a technique presented in [9, 10], which

¹Note that this landmark pattern already includes the additional ordering previously discussed; the patient is checked for being brain death before the permission is obtained.

was originally developed with model-checking in mind (some model-checkers, like SPIN, [6], are built around principles very similar to those of this technique). The idea is that we create a generic, canonical-like model representing *all* LTL models that satisfy the norms of the system. This canonical model is, in fact, a finite state machine as we show later. From this finite state machine we generate a *regular expression* expressing the characteristic features of all models satisfying the norms. We will show that this expression is, in fact, a basic landmark pattern, exactly containing all the important states (landmarks) expressed in the norms, as well as the order in which these states must occur (thus making it a landmark pattern). This normative pattern will then be expanded with extra landmarks to strengthen it to a full landmark pattern (as described above). The landmark pattern, now including all important states, both normative-wise and efficiency-wise, will then be translated into a protocol for the normative domain.²

Norms are expressed in linear-time temporal logic (similar to []). The temporal logic that we are going to use is the following (note that any LTL logic can be used for this, as long as the formulas are in negated normal form). The formulas of linear-time temporal logic built from a set of atomic propositions are the following:

- **true**, **false**, p , $\neg p$ for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are LTL formulas;
- $\bigcirc \varphi_1$, φ_1 **until** φ_2 , and φ_1 **releases** φ_2 , where φ_1 and φ_2 are LTL formulas.

The operator \bigcirc is a next-state operator, denoting that φ_1 must hold in the state following the present one. The **until** operator is a strong until, denoting that φ_1 must hold in all states up to a state where φ_2 holds (which should appear). The **release** operator is the dual of **until** and requires that φ_2 is always true, a requirement that is released as soon as φ_1 holds. The *sometime* ($\diamond \varphi_1$) and *always* ($\square \varphi_1$) operators, denoting that φ_1 holds somewhere in the future or in every state from now on, respectively, are introduced as the following abbreviations: $\diamond \varphi_1 \equiv \mathbf{true\ until\ } \varphi_1$ and $\square \varphi_1 \equiv \mathbf{false\ releases\ } \varphi_1$.

The semantics of this logics, with respect to sequences $\sigma : \mathbb{N} \rightarrow 2^P$, where we write $\sigma(i)$ to denote the i^{th} state of σ and σ^j to denote the suffix of σ obtained through removing the first j states of σ (i.e. $\sigma^j(i) = \sigma(j+i)$), is given by the following rules:

- For all σ , we have $\sigma \models \mathbf{true}$ and $\sigma \not\models \mathbf{false}$;
- $\sigma \models p$ for $p \in P$ iff $p \in \sigma(0)$;
- $\sigma \models \neg p$ for $p \in P$ iff $p \notin \sigma(0)$;
- $\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$;

²Note that the technique described here is not designed to generate all possible protocols, but to provide help in the creation of protocols for a normative domain, in general. Generating all protocols from a landmark pattern would be very hard due to the subjective choices that can be made. The idea of our approach is more to give a “skeleton” or “prototypical” protocol by means of the landmark patterns, since, at least, the normative landmarks extracted from the norms must, in some way, be satisfied in every protocol for that task in a given domain.

- $\sigma \models \varphi_1 \vee \varphi_2$ iff $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$;
- $\sigma \models \bigcirc \varphi_1$ iff $\sigma^1 \models \varphi_1$;
- $\sigma \models \varphi_1$ **until** φ_2 iff there exists $i \geq 0$ such that $\sigma^i \models \varphi_2$ and for all $0 \leq j < i$, we have $\sigma^j \models \varphi_1$;
- $\sigma \models \varphi_1$ **releases** φ_2 iff for all $i \geq 0$ such that $\sigma^i \not\models \varphi_2$, there exists $0 \leq j < i$ such that $\sigma^j \models \varphi_1$.

Using this logic we can, as done in [], give a representation of the normative operators that we are going to use to represent the norms. We limit ourselves to obligations, which can be expressed in the logic presented above as:

$$O(\rho < \delta) \Leftrightarrow \diamond \delta \wedge \left[(\neg \delta \wedge \neg \rho \wedge \neg v(\rho, \delta)) \mathbf{until} \right. \\ \left. ((\rho \wedge \neg \delta \wedge \square \neg v(\rho, \delta)) \vee (\neg \rho \wedge \delta \wedge \bigcirc v(\rho, \delta))) \right]$$

Although prohibitions also play an important role in normative specifications we do not treat them separately here as they can be expressed in terms of obligations. I.e.

$$F(\rho) \Leftrightarrow \square O(\neg \rho < \mathbf{true})$$

As mentioned before, the landmarks will be extracted from an automaton that is generated on basis of the LTL specification of the norms.

The automata that we consider are Büchi automata on infinite words. Infinite words are sequences of symbols isomorphic to the natural numbers, i.e. a mapping from the infinite sequence to symbols of the alphabet Σ ; $w : \mathbb{N} \rightarrow \Sigma$. Büchi automata have exactly the same structure as traditional finite word automata, with the exception that their semantics are defined over infinite words. A Büchi automaton is defined as a tuple $A = (\Sigma, S, \Delta, S_0, \mathcal{F})$ where

- Σ is an alphabet,
- S is a set of states,
- $\Delta : S \times \Sigma \rightarrow S$ (deterministic) or $\Delta : S \times \Sigma \rightarrow 2^S$ (nondeterministic) is a transition function,
- $S_0 \subseteq S$ is a set of initial states (a singleton for deterministic automata), and
- $\mathcal{F} = \{F_1, \dots, F_k\}$, a set of sets of accepting states where $F_i \subseteq S$ for every $F_i \in \mathcal{F}$

A word w is accepted (or recognised) by A if there exists a sequence $\lambda : \mathbb{N} \rightarrow S$ of states such that

- $\lambda(0) \in S_0$ (the initial state of λ is an initial state of A),
- $\forall 0 \leq i, \lambda(i+1) \in \Delta(\lambda(i), w(i))$ (the sequence of states is compatible with the transition relation of A),
- For each $F_i \in \mathcal{F}$, $\text{inf}(\lambda) \cap F_i \neq \emptyset$ where $\text{inf}(\lambda)$ is the set of states that appear infinitely often in τ (the set of repeating states of λ intersects the accepting set F).

Given these definitions of LTL and Büchi automata we can now present the relation between the logic and the automata and the procedure to create an automaton that accepts exactly those sequences that satisfy a formula φ . The translation from LTL formulas to Büchi automata is taken

from work presented in [9, 10]. The idea is that the sequences accepted by an automaton correspond exactly to those LTL sequences satisfying the formula.

A state s in the sequence σ is an LTL state characterised in the propositional elements that hold in that LTL state. These LTL state descriptions will ultimately form the symbols of the language accepted by the automaton, the sequence of these state descriptions being the words accepted by the automaton. A closure labelling τ is defined to denote the temporal formulas that must *at least* hold at the different stages of an automaton run, they will ultimately be used as the labellings of the automaton states. This labelling τ of a sequence σ indicates which (temporal) subformulas of φ hold at each state of σ , i.e. a subformula φ_1 of φ labels a position i (written as $\varphi_1 \in \tau(i)$), if and only if $\sigma_i \models \varphi_1$. We define the set of subformulas of a formula φ , called the *closure* of φ ($cl(\varphi)$), as follows:

- $\varphi \in cl(\varphi)$;
- $\varphi_1 \wedge \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$;
- $\varphi_1 \vee \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$;
- $\bigcirc \varphi_1 \in cl(\varphi) \Rightarrow \varphi_1 \in cl(\varphi)$;
- $\varphi_1 \text{ **until** } \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$;
- $\varphi_1 \text{ **releases** } \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$.

The closure labelling τ of a sequence σ , denoting the formulas of the closure of φ that hold at a given position, is then defined in such a way that it guarantees the correspondence between the positions in τ with positions in σ ; i.e. the closure labelling is defined by means of a set of rules that mirror LTL semantics. The closure labelling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ of a sequence $\sigma : \mathbb{N} \rightarrow 2^P$ for a formula φ over a set of atomic propositions P is valid when it satisfies the following rules for every $i \geq 0$:

1. **falsum** $\notin \tau(i)$;
2. for $p \in P$, if $p \in \tau(i)$ then $p \in \sigma(i)$, and if $\neg p \in \tau(i)$ then $p \notin \sigma(i)$;
3. if $\varphi_1 \wedge \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ and $\varphi_2 \in \tau(i)$;
4. if $\varphi_1 \vee \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ or $\varphi_2 \in \tau(i)$;

These rules ensure that the propositional part of LTL is satisfied by the closure labelling. Note that because the rules are specified as “if” rules and not as “if and only if” rules, the closure labelling is not required to be *maximal*, i.e. the rules give the requirements that *must* be satisfied by the closure labelling, but do not require that all formulas of the closure that hold at a given position are included in the label of that position.

For the temporal operators, the following rules are given.

5. if $\bigcirc \varphi_1 \in \tau(i)$ then $\varphi_1 \in \tau(i+1)$;
6. if $\varphi_1 \text{ **until** } \varphi_2 \in \tau(i)$ then either $\varphi_2 \in \tau(i)$, or $\varphi_1 \in \tau(i)$ and $\varphi_1 \text{ **until** } \varphi_2 \in \tau(i+1)$;
7. if $\varphi_1 \text{ **releases** } \varphi_2 \in \tau(i)$ then $\varphi_2 \in \tau(i)$, and either $\varphi_1 \in \tau(i)$ or $\varphi_1 \text{ **releases** } \varphi_2 \in \tau(i)$.

Rule 5 for the \bigcirc operator follows directly from the LTL semantics of the operator. For the **until** and **releases** operators, however, the equivalences $\varphi_1 \text{ **until** } \varphi_2 \equiv (\varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \text{ **until** } \varphi_2)))$ and $\varphi_1 \text{ **releases** } \varphi_2 \equiv (\varphi_2 \wedge (\varphi_1 \vee \bigcirc(\varphi_1 \text{ **releases** } \varphi_2)))$ are used for the definition of rules 6 and 7 instead, since they avoid the reference to a possibly infinite set of points in the sequence (it can be easily shown that these equivalences follow from the semantics of these operators).

Unfortunately, an extra rule is required to guarantee that the labelling satisfies subformulas with an **until**. Since rule 6 does not force the existence of a point at which φ_2 appears, this can be postponed forever (which is inconsistent to the LTL semantics of the **until** operator). An extra rule to guarantee the existence of this point satisfying the *eventuality* (formulas of the form $\varphi_1 \text{ **until** } \varphi_2$ are called eventualities, because φ_2 must eventually hold) has to be added:

8. if $\varphi_1 \text{ **until** } \varphi_2 \in \tau(i)$ then there is a $j \geq i$ such that $\varphi_2 \in \tau(j)$.

Given these restrictions, a formalisation of the relation between the closure labelling and the LTL sequence is given in [10]:³

THEOREM 1. *Consider a formula φ defined over a set of propositions P and a sequence $\sigma : \mathbb{N} \rightarrow 2^P$. One then has that $\sigma \models \varphi$ iff there is a closure labelling $\tau : \mathbb{N} \rightarrow 2^{cl(\varphi)}$ of σ satisfying rules 1-8 and such that $\varphi \in \tau(0)$.*

Given this theorem, the relation between an automaton A accepting all sequences satisfying φ and the LTL models is rather obvious. Recall that automata accept infinite sequences (words) when this sequence can be labelled by states of the automaton, while satisfying the conditions that the first state of the sequence is a start state of A , the transition relation of A is respected and the acceptance condition of A is met. It then becomes obvious to use $2^{cl(\varphi)}$ as state set (and possible state labels), and create an automaton over the alphabet 2^P that satisfies the necessary properties expressed in the labelling rules expressed above.⁴

DEFINITION 2. *A Büchi automaton for a formula φ is a tuple $A_\varphi = (\Sigma, S, \Delta, S_0, \mathcal{F})$, where*

- $\Sigma = 2^P$
- S is the set of states $\mathbf{s} \subseteq 2^{cl(\varphi)}$ that satisfy
 - **falsum** $\notin \mathbf{s}$;
 - if $\varphi_1 \wedge \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ and $\varphi_2 \in \mathbf{s}$;
 - if $\varphi_1 \vee \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ or $\varphi_2 \in \mathbf{s}$.
- The transition function Δ is defined as $\mathbf{t} \in \Delta(\mathbf{s}, \mathbf{a})$ iff
 - For all $p \in P$, if $p \in \mathbf{s}$ then $p \in \mathbf{a}$.

³The proof of this theorem can be found in [10].

⁴The alphabet of an automaton A for a formula φ consists of all possible LTL-worlds, being that an LTL-world is described as the collection of the propositions that hold in that world. For instance, if we only consider the propositions ρ, δ and γ , the LTL-world that satisfies only ρ and δ will be denoted by $\{\rho, \delta\}$ (we also use $\rho\delta$ or $\delta\rho$ to denote this world). Conversely, the label γ denotes the LTL world in which γ holds, but ρ and δ are false.

- For all $p \in P$, if $\neg p \in \mathbf{s}$ then $p \notin \mathbf{a}$.
- If $\bigcirc \varphi_1 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{t}$.
- If φ_1 **until** $\varphi_2 \in \mathbf{s}$ then either $\varphi_2 \in \mathbf{s}$, or $\varphi_1 \in \mathbf{s}$ and φ_1 **until** $\varphi_2 \in \mathbf{t}$.
- If φ_1 **unless** $\varphi_2 \in \mathbf{s}$ then $\varphi_2 \in \mathbf{s}$ and either $\varphi_1 \in \mathbf{s}$, or φ_1 **unless** $\varphi_2 \in \mathbf{t}$.

- $S_0 = \{\mathbf{s} \in S \mid \varphi \in \mathbf{s}\}$.

PROPOSITION 1. The Büchi automaton A_φ from definition 2 accepts all and only the sequences $\sigma : \mathbb{N} \rightarrow 2^P$ that satisfy the formula φ .

The restriction on the states S of A_φ ensures that the states (and thus the closure labels) satisfy rule 1 as well as rules 3 and 4 specified above. Rule 2 and rules 5-7 are enforced in the transition function Δ of A_φ . This ensures that the automaton A_φ complies to the rules 1-7 of the closure labelling specified above. The restriction on the start states S_0 of A_φ ensures that φ appears in the label of the first position of the sequence (thus limiting the labellings of A_φ to those where $\varphi \in \tau(0)$, as required by theorem 1). To ensure rule 8, the acceptance condition of the automaton is used. Rule 8 specifies that every state that contains an eventuality $e(\varphi')$ (where φ_1 **until** $\varphi' \in cl(\varphi) \Rightarrow e(\varphi') \in cl(\varphi)$) is followed at some point by a state that contains φ' . This means that labellings where $e(\varphi')$ appears indefinitely without φ' ever appearing must be avoided. These labellings can be avoided by requiring that the automaton goes infinitely often through a state in which both $e(\varphi')$ and φ' appear or in which $e(\varphi')$ does not hold. To achieve this, the acceptance condition of A_φ is specified as the following generalised Büchi condition:

- If the eventualities appearing in $cl(\varphi)$ are $e_1(\varphi_1), \dots, e_m(\varphi_m)$ then $\mathcal{F} = \{F_1, \dots, F_m\}$ where $F_i = \{\mathbf{s} \in S \mid e_i \varphi_i \in \mathbf{s} \vee e_i \notin \mathbf{s}\}$.

In accordance to this definition of an automaton accepting the sequences that satisfy φ , [10] states a procedure to generate a minimal Büchi automaton satisfying the constraints mentioned.

Using the relation between LTL and automata presented above we can now create an automaton that models all LTL sequences that satisfy the norms of a given domain. The norms, expressed as LTL formulas as presented earlier in this section, form the basis of the formula φ of which the automaton is built. Lets consider, for illustrative purposes, a domain governed by a single deadline $O(\rho < \delta)$ (ρ needs to happen one or more states before δ appears). If a protocol needs to be created for this domain, we need to extract the landmarks specified by the norms governing the domain. These landmarks, as explained earlier, are the characteristic features of the norms that define the basic structure of protocols that comply to that norms. The landmarks are extracted from the norms by creating an automaton (which, in fact, is a general, canonical-like model of the norms, since it represents all LTL sequences satisfying that set of norms) by means of the procedure described above. However, building an automaton on basis of the logical representation of the norms gives a model of the norms that also includes the LTL sequences where (one of the) norms have been violated, since the violation of a norm is just as much a part of the logical representation because of its prescriptive

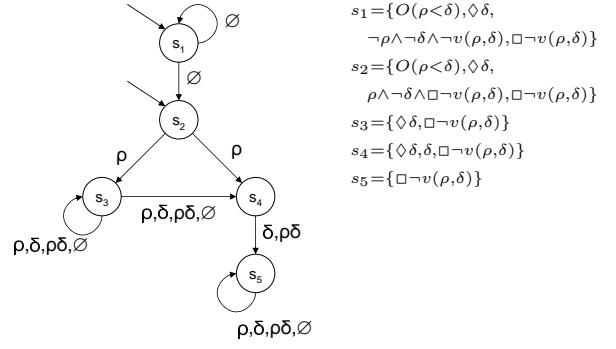


Figure 2: A Büchi automaton for $O(\rho < \delta)$.

nature (norms express what should be, not what is). Instead, we are more interested in only those LTL sequences where the norms hold but are not violated, since these sequences characterise the patterns that we want to capture in the protocol that we are creating. In case of our example, this means we need to build an automaton for the formula $O(\rho < \delta) \wedge \square \neg v(\rho, \delta)$ ⁵. The resulting automaton A_φ , generated by the protocol described above, is shown in figure 2. The alphabet of the automaton is taken as $\Sigma_\varphi = 2^{\{\rho, \delta, v(\rho, \delta)\}}$, the states $s_1, \dots, s_5 \subseteq 2^{cl(\varphi)}$ (parts of the state labels are given in figure 2), the starting states of the automaton are s_1 and s_2 , and the acceptance set is defined as $\mathcal{F}_\varphi = \{\{s_4, s_5\}, \{s_2, s_4, s_5\}\}$.

As can be seen, this automaton exactly represents our intuition of a deadline (as expressed in LTL logics). All LTL sequences satisfying a deadline, should have a number of states (possibly zero) in which nothing interesting happens (this is represented in state s_1). Then a state occurs where ρ holds (state s_2), after which, one or more states later, δ holds (the intermediate states are represented in state s_3 , the state where δ occurs is represented in state s_4). After the obligation has been fulfilled (δ has happened, while ρ happened one or more states before δ), an infinite sequence of states occurs where anything can happen as long as $v(\rho, \delta)$ does not happen; this is represented in state s_5 (since no more restrictions are posed on ρ and δ , they can hold in any order at any state after the deadline has been fulfilled). Note that, as required, none of the states satisfies $v(\rho, \delta)$. To fulfil the Büchi acceptance condition the sequences before ρ has happened (i.e. the transition from s_1 to s_1), and the sequence before δ happens (i.e. the transition from s_3 to s_3), can only be of finite length, the only infinite recursion in this automaton is the transition from s_5 to itself.

After translating the norms to a Büchi automaton we can extract a regular expression characterising the language expressed by the automaton. The idea is that the main characteristics, the basic landmark structure, obtained from the norms that are represented in the Büchi automaton, can be easily extracted through this regular expression.

Büchi automata accept a specific type of regular languages, namely ω -regular languages (or ω -languages, see [8, 7]) The

⁵Note that building an automaton for this formula is actually the same as building an automaton for a simplified representation of an obligation that cannot be violated: $\diamond \delta \wedge [(\neg \delta \wedge \neg \rho \wedge \neg v) \text{ until } (\rho \wedge \neg \delta \wedge \square \neg v)]$.

major difference between ω -languages and regular languages is, like the difference between Büchi automata and normal finite automata, that ω -languages are composed of infinite sequences (ω -words), where regular languages only contain finite words (the sequences are of finite length). The following property of Büchi automata, taken from [2] then expresses the language recognised by a Büchi automaton A .

THEOREM 2. *An ω -language $L_\omega \subseteq \Sigma^\omega$ is Büchi recognisable iff L is a finite union of sets $U.V^\omega$ where $U, V \subseteq \Sigma^*$ are regular sets of finite words.*

The representation of an ω -language in the form $L_\omega = \bigcup_{i=1}^n U_i.V_i^\omega$, where U_i, V_i are given by regular expressions, is called an ω -regular expression.

For the automaton presented in figure 2, we can express the accepted language by the following ω -regular expression

$$L_\omega(A_\varphi) = \emptyset^*.\rho.all^*.(\delta + \rho\delta).all^\omega.$$

As can be easily seen from this ω -regular expression, the points of interest of every LTL sequence satisfying φ are the state where ρ holds and the state where either δ or $\rho\delta$ holds (the former being the state where $\delta \wedge \neg\rho \wedge \neg v(\rho, \delta)$ holds, the latter where $\delta \wedge \rho \wedge \neg v(\rho, \delta)$ holds). As seen in the expression, confirming the intuition about deadlines, all LTL sequences satisfying $O(\rho < \delta)$ have a state where ρ holds (possibly preceded by a number of states where neither ρ nor δ hold), which always happens before a state occurs where δ holds (the ω -regular expression allows a number of states, possibly zero, between the occurrence of ρ and δ , in which ρ may occur as well). The fact that the state where δ should hold is denoted in the expression as $\delta + \rho\delta$ is mainly because, since the restriction of ρ at least happening before δ has already been met, at this point it does not really matter whether ρ holds or not (basically, the transition label $\delta + \rho\delta$ expresses no information concerning ρ or $\neg\rho$, but expresses that *at least* δ holds).

Given that we can deduce from the label $\delta + \rho\delta$ that at least δ holds, we can simplify the regular expression to the following landmark pattern

$$\mathcal{L}_\varphi = \langle \{\rho, \delta\}, \emptyset, \{(\rho, \delta)\} \rangle$$

This is the normative landmark pattern extracted from the norms of the domain (just $O(\rho < \delta)$ in this case). This landmark pattern is the basis of the landmark pattern that we construct to create protocols. The landmark pattern will now be extended with additional landmarks to denote domain-specific information and information to increase the efficiency and feasibility of the pattern.

4. STRENGTHENING THE PATTERN

In section 2 we presented the idea of using landmarks to bridge the gap between norms and practise (in this particular case, the norms and the protocols). Part of this idea to create the bridge, taken from the relation between norms and practise in the real world, was to extend the landmarks extracted from the norms with additional information. We have shown above how the basic landmark pattern, the normative landmark pattern, can be extracted automatically from the norms. Similar to what we explained in section 2, we will use this “skeleton” landmark pattern and extend it with additional landmarks (and orderings between landmarks) to create a landmark pattern that also incorporates,

next to the restrictions on the protocol given by the norms, information about efficiency and feasibility that is normally not included in the normative specification of the domain, but rather taken from practise itself.

This would mean that the landmark pattern for our example domain, \mathcal{L}_φ , is extended with additional landmarks and orderings. The additional landmarks are, in fact, filling in the “gaps” between the normative landmarks in the ω -regular expression of this normative domain, $L_\omega(A) = \emptyset^*.\rho.all^*.(\delta + \rho\delta).all^\omega$. We will show that adding these additional landmarks creates a new ω -regular expression that represents a language L'_ω that is accepted by A_φ if the alphabet of A_φ is extended accordingly. Note, though, that this does not count as a one-one relation, as the adapted automaton accepts more words than just those expressed in L'_ω (though still satisfying only the LTL sequences that satisfy φ). Meaning, the automaton A'_φ , which is the adaptation of A_φ , accepts words that are not included in L'_ω . We do not intend to build a new automaton that exactly matches the restrictions in the extended landmark pattern, but just to show that the new landmark pattern is an instantiation of the normative landmark pattern (i.e. the extended landmark pattern is a special case of the normative landmark pattern).

Let us assume that a landmark γ can be added to the normative landmark pattern, between ρ and δ , to increase efficiency. The landmark pattern resulting from this addition would be

$$\mathcal{L}'_\varphi = \langle \{\rho, \delta, \gamma\}, \emptyset, \{(\rho, \gamma), (\gamma, \delta), (\rho, \delta)\} \rangle.$$

By extending the ω -regular expression from the normative landmark pattern in a similar way, we can show that the (slightly changed) automaton A'_φ accepts the words represented by this pattern, thus reinforcing that the extended landmark pattern is still compliant to the norms. This extended ω -regular expression is the following

$$L'_\omega(A'_\varphi) = \emptyset^*.\rho.(\emptyset + \rho + \gamma + \rho\gamma)^*.(\gamma + \rho\gamma).all^*.(\delta + \rho\delta).all^\omega.^6$$

The change to the automaton A_φ to create A'_φ that accepts words from this language, is merely the addition of γ to the alphabet (if the alphabet of the automaton does not contain γ it can never accept words that include γ). This change also means that the labels of the transitions of the automaton are changed. Basically, we re-run the procedure presented above to create an automaton for φ over an alphabet $\Sigma'_\varphi = 2^{\{\rho, \delta, \gamma, v(\rho, \delta)\}}$, however, since we did not add any information to the formula φ that has to be satisfied by the sequences of A'_φ , nothing changes in the structure of A'_φ (no new states or transitions are added), we only added a proposition of which no restrictions are given (i.e. γ can be true or false in any state of the LTL sequences, or, no information about the truthvalue of γ is given in any of the states of the automaton).

The relations that the additional landmarks express are not incorporated in the automaton A'_φ and if one would want these restrictions to be expressed in A'_φ , the procedure presented above can be run on an adapted formula φ' that expresses, next to the norms, these restrictions. However, for the purpose that we expressed in section 2, it is enough

⁶Note that the first intuition of this ω -regular expression, being $\emptyset^*.\rho.all^*.\gamma.all^*.(\delta + \rho\delta).all^\omega$, is not correct, due to the fact that it allows δ to appear before γ appears (remember that $all = (\emptyset + \rho + \delta + \rho\delta)$).

to proceed as presented above, i.e. only check whether the extended landmark pattern is still compliant to the norms. Remember that, next to the restriction that the extended landmark pattern should remain norm-compliant, the extended landmark pattern must satisfy the reachability and capability constraints specified in section 2.

4.1 Landmarks to Protocols

Given that the landmark pattern expresses states that should be achieved, it can be viewed as a collection of concrete goals and the order in which these goals must be achieved. A translation from a landmark pattern to a basic, prototypical protocol is then achieved by means of use of *seeing to it that* operators (*stit*) [1]. While the *stit* operator ignores the means by which an agent will bring about a state of affairs, it does provide the link to make states (the landmarks) into procedural goals. It is then possible to create a protocol given this specification of goals (while retaining the order in which they should be achieved) by linking these goals to the capabilities of agents via a planning algorithm, e.g. like STRIPS [5].

Let us illustrate this by means of our example, where it means that the landmark pattern $\mathcal{L}'_\rho = \langle \{\rho, \delta, \gamma\}, \{\rho \leq \gamma, \gamma \leq \delta, \rho \leq \delta\} \rangle$ is translated to an ordering of goals, represented as a sequence of (still abstract) actions: (*stit* ρ); (*stit* γ); (*stit* δ). Thus expressing that it should be the case that ρ is achieved first, then γ and finally δ .⁷ The capabilities of the agents in the domain can then be used to expand this action sequence composed of abstract actions (which abstracts from the means necessary to achieve the expressed goals) to a full protocol (again, expressed as a sequence of actions). For this example, let us assume that the agents have the following capabilities (we use a ‘‘STRIPS-like’’ representation of the agents capabilities, by expressing the necessary pre-conditions and post-conditions of the actions, such a definition of an agent’s capabilities is found in, for example, the 3APL agent programming language, [3]):

$$\begin{aligned} Op(\text{Action} : \text{action}_1, \text{Effect} : \rho) \\ Op(\text{Action} : \text{action}_2, \text{Precond} : \rho, \text{Effect} : \eta) \\ Op(\text{Action} : \text{action}_3, \text{Precond} : \eta, \text{Effect} : \gamma) \\ Op(\text{Action} : \text{action}_4, \text{Precond} : \gamma, \text{Effect} : \delta) \end{aligned}$$

Using the agent’s capabilities, we can use the planning algorithm to expand the abstract protocol (*stit* ρ); (*stit* γ); (*stit* δ) to the full protocol *action*₁; *action*₂; *action*₃; *action*₄.

As hinted in section 2, it might be possible that a landmark expresses a state that is not reachable by a single agent alone; a cooperation of agents might be needed to achieve (parts of) the landmark pattern. In this case, methods for designing interaction patterns between agents, such as the ones in OPERA, [4], can be applied to create the necessary interaction protocols for reaching those complex goals.

5. EXAMPLE

Let us look at an example to show the entire process described in the previous sections. This process starts with the

⁷Actually, due to δ being the deadline expressed in the norm, it is not necessarily a goal, but can just as well be an event that just happens. For this example we assume that δ is the goal of the task as a whole, and the norm is just describing that something must be done before that task can be completed.

LTL representation of the norms, and ends with a protocol (for a specific task) that is compliant to these norms.

The example we use here is based on the domain of organ transplantation. The task at hand, that should be achieved by the protocol, is to assign organs that have just become available (a suitable donor has been found, i.e. a patient who made a statement to permit post-mortem liver transplantation has died). This task, *assign_organ*, is the goal of the protocol that has to be achieved while keeping in mind the restrictions given by the norms of the domain. We assume that there exist one norm for this task (from the Dutch law on organ transplantations):

Before an organ is removed, death is certified by a professional doctor in accordance with the latest medically valid methods and criteria for determining brain death.

We represent this norm as the following (LTL) formula:

$$O(\text{certify_death} < \text{remove_organ})$$

Naturally, there exist a precedence ordering between the *remove_organ* state and the *assign_organ* state (one cannot assign organs before extraction). Since we use the *assign_organ* state as the ultimate goal state, we need to model this ordering as well:

$$\neg \text{assign_organ} \text{ until } \text{remove_organ} \wedge \diamond \text{assign_organ}$$

This formula expresses that $\neg \text{assign_organ}$ necessarily holds up to the point at which *remove_organ* holds, after which, at some moment, *assign_organ* will hold. Combined with the norm specified above, this gives us the LTL formula ψ that restricts the domain, and needs to be converted to a landmark pattern:

$$\begin{aligned} \psi \equiv O(\text{certify_death} < \text{remove_organ}) \wedge \\ \neg \text{assign_organ} \text{ until } \text{remove_organ} \wedge \diamond \text{assign_organ} \end{aligned}$$

The Büchi automaton A_ψ resulting from the translation of this set of norms is shown in figure 3 below (we abbreviate *remove_organ* to ρ , *assign_organ* to α and *certify_death* to γ). The most important elements of the state labels are the following:

$$\begin{aligned} s_1 &= \{O(\gamma < \rho), \neg \alpha \text{ until } \rho, \diamond \rho, \diamond \alpha, \neg \gamma \wedge \neg \rho \wedge \neg v(\gamma, \rho), \neg \alpha\} \\ s_2 &= \{O(\gamma < \rho, \neg \alpha \text{ until } \rho, \diamond \rho, \diamond \alpha, \gamma \wedge \neg \rho \wedge \square \neg v(\gamma, \rho), \neg \alpha\} \\ s_3 &= \{\diamond \rho, \diamond \alpha, \square \neg v(\gamma, \rho), \neg \alpha\} \\ s_4 &= \{\diamond \rho, \diamond \alpha, \square \neg v(\gamma, \rho), \rho, \neg \alpha\} \\ s_5 &= \{\diamond \alpha, \square \neg v(\gamma, \rho)\} \\ s_6 &= \{\diamond \alpha, \square \neg v(\gamma, \rho), \alpha\} \\ s_7 &= \{\square \neg v(\gamma, \rho)\} \end{aligned}$$

The acceptance condition of A_ψ is given as the set $\mathcal{F}_\psi = \{\{s_6, s_7\}, \{s_4, s_5, s_6, s_7\}, \{s_2, s_4, s_5, s_6, s_7\}\}$.

Using the automaton of figure 3 we create an ω -regular expression to describe the language accepted by the automaton. This ω -regular expression captures all the important features of the LTL structures of ψ that are represented in A_ψ . We abbreviate $(\rho + \gamma + \alpha + \emptyset + \rho\gamma + \alpha\rho + \alpha\gamma + \alpha\rho\gamma)$ to *all* (it denotes that everything but $v(\gamma, \rho)$ can hold at that moment). Note that the expression allows for multiple occurrences of γ , ρ and α (although they need to occur in a particular order), i.e. words like $\rho.\gamma\rho.\alpha\rho.\alpha^\omega$ are accepted by the automaton, while, in practise, only one occurrence of each of these states is more likely; e.g. after the death of the patient has been certified, it will not have to be done

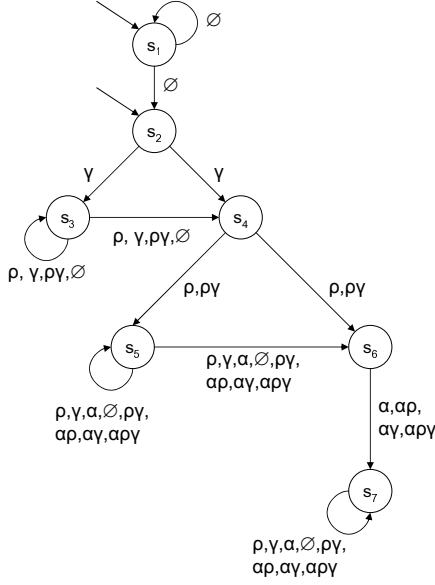


Figure 3: Example Automaton.

again (for that same patient). Basically, the practise that we are trying to model is best described by the ω -expression: $\emptyset^*.\gamma.\emptyset^*.\rho.\emptyset^*.\alpha.\emptyset^\omega$. This, however, is not a weakness of the technique we are describing here, but merely a weakness of the LTL representation that we used. Since we have not specified in our LTL representation that the states will only happen once, they *can* happen multiple times. The LTL representation only models the restrictions on the ordering of the state occurrences.

$$\emptyset^*.\gamma.(\rho+\gamma+\rho\gamma+\emptyset)^*.(\rho+\rho\gamma).all^*.(\alpha+\alpha\rho+\alpha\gamma+\alpha\gamma\rho).all^\omega$$

From this ω -regular expression we can derive the following (normative) landmark pattern (remember that $\rho+\gamma\rho$ denotes that *at least* ρ should hold, and that $\alpha+\alpha\rho+\alpha\gamma+\alpha\gamma\rho$ denotes that *at least* α holds):

$$\mathcal{L}'_\psi = \langle \{\gamma, \rho, \alpha\}, \{\gamma \leq \rho, \rho \leq \alpha\} \rangle$$

Using knowledge from the domain we strengthen the landmark pattern with additional landmarks. In this case, we use the knowledge that before the assignment the compatibility between the organ and the donor must be checked, e.g. to see if the blood type of the donor and the recipient match. To increase the efficiency of the assigning, we can check the blood of the donor when the death has been certified (*check_bloodtype*, shortened as β), giving us the following landmark pattern:

$$\mathcal{L}_\psi = \langle \{\gamma, \rho, \alpha, \beta\}, \{\gamma \leq \rho, \rho \leq \alpha, \gamma \leq \beta, \beta \leq \rho\} \rangle$$

The last steps of the process is converting the landmark pattern to a basic, prototypical protocol, using *stit* operators: *stit*(γ); *stit*(β); *stit*(ρ); *stit*(α), which is then translated, by using the capabilities of the agents, to a protocol:

execute_brain_death_protocol; *take_blood_sample*;
test_blood(bloodtype); *start_operation*; *remove_organ(liver)*;
assign_organ(patient, bloodtype); *operate(patient)*

This protocol can be used as a basic pattern for agents that have to fulfill the norm that we started out with. Of course

it can still be extended with extra actions to provide for specific situations.

6. CONCLUSIONS

In this paper we have described a procedure to derive a basic protocol from norms described as obligations with deadlines. This procedure can be used to provide protocols for e-institutions governed by norms such that agents that follow these protocols will always fulfill all the norms of the e-institution.

We have not shown in this paper how the process can be conducted using multiple norms. Basically, this process is a very simple extension of the above described procedure. The different norms are all described in LTL. We can combine the norms in LTL by just taking the conjunction of them. If more knowledge is available on the relation between the norms that is not explicit in the norms themselves this can also be added in the LTL description. Often this amounts to temporal orderings between deadlines of the norms that derive from common-sense knowledge. E.g. asking consent for organ donation from family of a donor should be done after the donor has died (not before). The resulting formula can then be processed as before again.

Another point that we could not expose in depth due to space limitations is the construction of a protocol involving multiple parties. Mainly what comes out of the procedure above is a set of states that should be reached by the agents. If a state can only be reached by a coordinated action of several agents we can use techniques from MAS planning to create an interaction pattern to reach that state. Although not at all trivial we assume that existing techniques suffice to bridge this gap.

Finally, we have not formally shown that the resulting protocols indeed satisfy the norms. This can be done by verifying that the protocol will never lead to a violation state. In [?] we have shown that it is indeed possible to perform this exercise, therewith ensuring the correctness of the complete procedure.

7. REFERENCES

- [1] N. Belnap and M. Perloff. Seeing to it that: a canonical form for agentives. *Theoria*, 54:175–199, 1988.
- [2] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Conference on Logic, Methodology and Philosophy of Science*, pages 1–11, Stanford, CA, 1962. Stanford University Press.
- [3] M. Dastani, B. van Riemsdijk, and J.-J. Ch. Meyer. Programming multi-agent systems in 3apl. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.
- [4] V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic*. SIKS Dissertation Series 2004-1. SIKS, 2004. PhD Thesis.
- [5] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.
- [6] G. Holzmann. The model checker spin. *IEEE Transactions On Software Engineering*, 23(5):279,

- 1997.
- [7] L. Staiger. ω -languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 339–387. Springer-Verlag, Berlin, 1997.
 - [8] W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.
 - [9] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
 - [10] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. In *Lectures on Formal Methods and Performance Analysis*, number 2090 in LNCS, pages 261–277. Springer-Verlag, New York, 2002.