Checking Traces for Regulatory Conformance*

Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky

Department of Computer Science
University of Pennsylvania
Philadelphia, PA 19104-6389, USA
{nikhild,joshi,lee,sokolsky}@seas.upenn.edu

Abstract. We consider the problem of checking whether the operations of an organization conform to a body of regulation. The immediate motivation comes from the analysis of the U.S. Food and Drug Administration regulations that apply to bloodbanks - organizations that collect, process, store, and use donations of blood and blood components. Statements in such regulations convey constraints on operations or sequences of operations that are performed by an organization. It is natural to express these constraints in a temporal logic.
There are two important features of regulatory texts that need to be accommodated by a representation in logic. First, the constraints conveyed by regulation can be obligatory (required) or permitted (optional). Second, statements in regulation refer to others for conditions or exceptions. An organization conforms to a body of regulation if and only if it satisfies all the obligations. However, permissions provide exceptions to obligations, indirectly affecting conformance.
In this paper, we extend linear temporal logic to distinguish between obligations and permissions, and to allow statements to refer to others. While the resulting logic allows for a direct representation of regulation, evaluating references between statements has high complexity. We discuss an empirically motivated assumption that lets us replace references with tests of lower complexity, leading to efficient trace-checking algorithms in practice.

1 Introduction

Regulations, laws and policies that affect many aspects of our lives are represented predominantly as documents in natural language. Mechanically checking compliance with these regulations and policies is an area of growing importance [1–3].

In this paper, we will consider one such regulation, the Food and Drug Administration's Code of Federal Regulations (FDA CFR) [4] that governs the operations of U.S. bloodbanks. The CFR is developed by experts in the field of medicine, and regulates the tests that need to be performed on donations of blood before they are used.

Bloodbanks are organizations that perform collection, testing, storage, and distribution of blood donations and are required to conform to the regulation (CFR). The operations of a bloodbank are logged in a database that keeps track of donations that are collected by the bloodbank, tests that are performed on them and, ultimately, the

* This research was supported in part by NSF CCF-0429948, ARO W911NF-05-1-0158, and ONR MURI N00014-04-1-0735.

way each donation is used. Our goal is to check in an efficient manner that the operations as recorded in the database are compliant with the CFR, and to raise an alarm if a non-compliant action is detected. To achieve this goal, we first need to settle on an approach to formalize regulatory documents, and then consider the feasibility of checking database logs with respect to the formalized regulations.

As we illustrate with examples in Section 2, the basic structure of regulatory statements is to declare that a certain action can take place when certain conditions apply. At a first glance, it seems that such statements can be encoded as logical clauses, where a set of preconditions imply a postcondition. However, there are two complications that need to be addressed. First, regulations convey permissions and obligations, which have to be reflected in the formal description and handled accordingly during the checking. Second, a common phenomenon in regulatory texts is for sentences to function as conditions or exceptions to others. This function of sentences makes them dependent on others for their interpretation, and makes the translation to logic difficult. We call this the problem of *references to other laws*, and it is the central focus of this paper.

In Section 2, we argue that a logic to represent regulation should provide mechanisms for statements to refer to others, and to make inferences from the sentences referred to. We then turn to formalization of regulatory documents and regulated operations. In Section 3.1, we define an abstract model for representing the operations of an organization, followed in Section 3.2 by a predicate-based linear temporal logic to express normative statements in regulation. Formal definitions of conformance are given. We then extend the logic to allow sentences to refer to others, in Section 3.3.

Section 4 describes the checking process. We adapt the methodology of the rule-based formalism Eagle [5] to handle references. In order to check statements with references, we need to compute a fixed point, propagating information between references from one statement to another until we get a consistent evaluation. The evaluation of references has high complexity. We identify a condition, motivated by a case study of the CFR, under which references can be replaced by tests of lower complexity. We also discuss a prototype checking tool.

Section 5 concludes with a discussion of future research directions and a survey of related work.

2 Motivation

In this section, we consider a representative sample of the CFR and argue that a logic to represent regulation should provide a mechanism for sentences to refer to others.
Example. Below we present shortened versions of sentences from the CFR Section 610.40, which we will use as a running example throughout the paper.

(1) Except as specified in (2), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(2) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

Statement (1) conveys an obligation to test donations of blood or blood component for Hepatitis B, and (2) conveys a permission not to test a donation of source plasma

(a blood component) for Hepatitis B. To assess an organization's conformance to (1) and (2), it suffices to check whether "all non-source plasma donations are tested for Hepatitis B". In other words, (1) and (2) imply the following obligation:

(3) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B.

There are a variety of logics in which one can capture the interpretation of (3), as needed for conformance. For example, in first-order logic, one can write $\forall x : (d(x) \wedge \neg sp(x)) \Rightarrow test(x)$, where $d(x)$ is true iff x is a donation, $sp(x)$ is true iff x is a source plasma donation, and $test(x)$ is true iff x is tested for Hepatitis B. Thus, to represent (1) and (2) formally, we inferred that they implied (3) and (3) could be represented more directly in a logic.

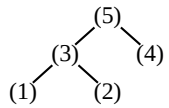
Now suppose we have a sentence that refers to (1):

(4) To test for Hepatitis B, you must use a screening test kit.

The reference is more indirect here, but the interpretation is: "if (1) requires a test, then the test must be performed using a screening test kit". A bloodbank is not prevented from using a different kind of test for source plasma donations. (4) can be represented by first producing (3), and then inferring that (3) and (4) imply the following:

(5) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B using a screening test kit.

It is easy to represent the interpretation of (5) directly in a logic. However, (5) has a complex relationship to the sentences from which it was derived, i.e., (1), (2) and (4). The derivation takes the form of a tree:



We argue that constructing a single derived obligation from multiple statements should be avoided. On the one hand, the derived obligation can become very complex. The full version of statement (1) in the CFR contains six exceptions, and these exceptions in turn have statements that qualify them further. A statement can be used as an exception to multiple other statements, and it is easy to see that the derived obligation can be exponentially larger than the original set of statements. We advocate an approach that allows us to introduce references into the syntax of the logic, and resolve references during evaluation.

3 Formalization of Regulatory Documents

In this section, we extend linear temporal logic (LTL) to distinguish between obligations and permissions, and allow references between statements. We begin, in Section 3.1, by representing a bloodbank as a run or trace. Section 3.2 extends LTL to distinguish between obligations and permissions, leading to definitions of conformance. We then extend the logic to allow sentences to refer to others (Section 3.3).

3.1 Model for Regulated Operations

Given the need to demonstrate conformance to the regulation in case of an audit, regulated organizations such as bloodbanks keep track of their operations in a database, for example donor information and the tests they perform. Such a system can be thought of abstractly as a relational structure evolving over time. At each point in time (state), there are a set of objects (such as donations and donors) and relations between the objects (such as an association between a donor and her donations). The state changes by the creation, removal or modification of objects. We represent this as a run.

Definition 1 (A Run of a System). *Given a set O (of objects) and countable sets $\Phi_1, ..., \Phi_n$ (where Φ_j is a set of predicate names of arity j), a run of a system $R(O, \Phi_1, ..., \Phi_n)$, abbreviated as R, is a tuple $(r, \pi_1, ..., \pi_n)$ where:*

- *$r : N \rightarrow S$ is a sequence of states. N is the set of natural numbers, and S is a set of states.*
- *$\pi_j : \Phi_j \times S \rightarrow 2^{O^j}$ is a truth assignment to predicates of arity j. Given $p \in \Phi_j$, we will say that $p(o_1, ..., o_j)$ is true at state s iff $(o_1, ..., o_j) \in \pi_j(p, s)$.*

Given a run R and a time $i \in N$, the pair (R, i) is called a point (statements in linear temporal logic are evaluated at points). Given the predicate names $(\Phi_1, ..., \Phi_n)$, the corresponding space of runs is denoted by $\mathcal{R}(\Phi_1, ..., \Phi_n)$, abbreviated as $\mathcal{R}$.

Time	Objects	Predicates
1	o_1	$d(o_1)$, $sp(o_1)$, $\neg test(o_1)$
2	o_1	$d(o_1)$, $sp(o_1)$, $\neg test(o_1)$
	o_2	$d(o_2)$, $\neg sp(o_2)$, $\neg test(o_2)$
3	o_1	$d(o_1)$, $sp(o_1)$, $test(o_1)$
	o_2	$d(o_2)$, $\neg sp(o_2)$, $\neg test(o_2)$

Table 1. A run of a bloodbank

Table 1 shows a possible run of a bloodbank. First, an object o_1 is entered into the system. o_1 is a donation of source plasma ($d(o_1)$ and $sp(o_1)$ are true). When a donation is added, its test predicate is initially false. Then, an object o_2 is added, which is a donation but not of source plasma. In the third step, the object o_1 is tested.

3.2 Logic for Regulatory Conformace

Predicate-based Linear Temporal Logic (PredLTL) The logic that we define in this section is a restricted fragment of first-order modal logic. The restriction is that we allow formulas with free variables, but no quantification over objects. Formulas will be interpreted using the universal generalization rule, i.e., over all assignments to free variables. The restrictions are similar in spirit to logic programs, which have been observed to be sufficiently expressive for the generic statements in regulation [6, 7].

Definition 2 (Syntax). *Given sets $\Phi_1, ..., \Phi_n$ (of predicate names) and a set of variables X, the language $L(\Phi_1, ..., \Phi_n, X)$, abbreviated as L, is the smallest set such that:*

- $p(y_1, ..., y_j) \in L$ *where* $p \in \Phi_j$ *and* $(y_1, ..., y_j) \in X^j$.
- *If* $\varphi \in L$, *then* $\neg\varphi \in L$ *and* $\Box\varphi \in L$. *If* $\varphi, \psi \in L$, *then* $\varphi \wedge \psi \in L$.

Disjunction $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$ and implication $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ are derived connectives. The temporal operator is understood in the usual way: $\Box\varphi$ (φ holds and will always hold (globally)). $\Diamond\varphi$ (φ will eventually hold) is defined as $\neg\Box\neg\varphi$.

We now extend the syntax to express normative statements in a body of regulation, by distinguishing between obligations and permissions.

Definition 3 (Syntax of Regulation). *Given a finite set of identifiers ID, a body of regulation Reg is a set of statements such that for each $id \in ID$, there exist $\varphi, \psi \in L$ such that either: id.***o***: $\varphi \rightsquigarrow \psi \in Reg$, or id.***p***: $\varphi \rightsquigarrow \psi \in Reg$*

id.**o**: $\varphi \rightsquigarrow \psi$ (id.**p**: $\varphi \rightsquigarrow \psi$) is read as: "it is obligated (permitted) that the precondition φ leads to the postcondition ψ".

Definition 4 (Semantics). *Given a run $R = (r, \pi_1, ..., \pi_n)$, $\varphi \in L$, and a variable assignment $v : X \to O$, the relation $(R, i, v) \models \varphi$ is defined inductively as follows:*

- $(R, i, v) \models p(y_1, ..., y_j)$ *iff* $(v(o_1), ..., v(o_j)) \in \pi_j(p, r(i))$.
- *The semantics of conjunction and negation is defined in the usual way.*
- $(R, i, v) \models \Box\varphi$ *iff for all* $k \geq i : (R, k, v) \models \varphi$.

We extend the semantic relation to regulatory staments. We take $\models$ to stand for "conforms to":

- $(R, i, v) \models$*id.***o***:* $\varphi \rightsquigarrow \psi$ *iff* $(R, i, v) \models \varphi \Rightarrow \psi$ *($\Rightarrow$ is implication)*
- $(R, i, v) \models$*id.***p***:* $\varphi \rightsquigarrow \psi$*. Runs vacuously conform to permissions. Permissions will become relevant when references from obligations are present (Section 3.3).*

Consider again our example from Section 2. We use three predicates defined as follows. $d(x)$ is true iff x is a donation. $sp(x)$ is true iff x consists of source plama. $test(x)$ is true iff x is tested for Hepatitis B.

Statement (3) is represented as: 3.**o**: $d(x) \wedge \neg sp(x) \rightsquigarrow \Diamond test(x)$. Statement (2) can be represented as: 2.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$. However, statement (1) cannot be represented directly.

The deontic concepts of obligation and permission are treated as properties of sentences. Only obligations matter for conformance. If a non-source plasma donation is not tested, there is a problem. On the other hand, a bloodbank may choose to test a donation of source plasma or not. In assessing conformance, the function of a permission is to serve as an exception to an obligation, and in this indirect manner it becomes relevant. We will give a semantics to this function of permissions in Section 3.3. Such a treatment of permissions has its basis in the legal theory of Ross [8].

In the formulation here, obligations and permissions are top-level operators and cannot be negated. This restriction can be removed in several ways, e.g., using a many-valued interpretation. However, we avoid this to simplify presentation. A more crucial restriction is that iterated deontic constructs cannot be expressed directly, i.e., sentences of the form "required to allow x" or "allowed to require x.". One has to decide what top-level obligations or permissions are implied by these constructs. To our knowledge, handling iterated constructs is an open problem in deontic logic [9].

Conformance of a run R is defined using the notion of validity. φ is valid at the point (R, i), $(R, i) \models \varphi$, iff for all variable assignments v: $(R, i, v) \models \varphi$. φ is valid in R, $R \models \varphi$ iff for all $i : (R, i) \models \varphi$.

Definition 5 (Run Conformance). *Given a body of regulation Reg and a run R representing the operations of an organization, we say that R conforms to the regulation iff for all obligations id.***o***: $\varphi \rightsquigarrow \psi \in Reg$ we have $R \models id$.***o***: $\varphi \rightsquigarrow \psi$.*

The definition of conformance is given in terms of obligations. We now extend the logic to allow sentences to refer to others making permissions relevant to conformance.

3.3 References to Other Laws

In this section, we describe the logical machinery we use to express and handle references to laws. We give an example-driven account here, followed by a formal account in the context of a runtime checking algorithm in Section 4.1.

We extend the syntax with *an inference predicate* $\mathrm{by}_{\mathrm{Id}}(\varphi)$, where Id is a set of identifiers. $\mathrm{by}_{\mathrm{Id}}(\varphi)$ is read as "by the laws in Id φ holds". There are two restrictions: (a) φ is a statement in PredLTL (Definition 2) and (b) the predicate $\mathrm{by}_{\mathrm{Id}}(\varphi)$ can appear only in preconditions of laws. These restrictions are similar to those that apply to justifications in default logic [11]. In the examples that we discuss, the set Id has a single element, i.e., a statement refers to a single other law. In general, laws refer to sets of statements, e.g., "except as specified in this section".

Consider again our example statements (1) and (2), which can now be represented as follows:

- 1.**o**: $d(x) \wedge \neg \mathrm{by}_{\{2\}}(\varphi(x)) \rightsquigarrow \Diamond test(x)$, and
- 2.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$

In the formula above, the subformula $\mathrm{by}_{\{2\}}(\varphi(x))$ is understood as "by the law (2) the formula $\varphi(x)$ holds". It remains to define the formula $\varphi(x)$. Intuitively, this should be the negation of the postcondition of (1). In other words, if $\neg\Diamond test(x)$ follows from (2), then the postcondition of (1) need not hold.[1]

1.**o**: $d(x) \wedge \neg \mathrm{by}_{\{2\}}(\neg\Diamond test(x)) \rightsquigarrow \Diamond test(x)$

We interpret $\mathrm{by}_{\{2\}}(\neg\Diamond test(x))$, by letting formulas have output. In other words, when the precondition of an obligation or permission is true at a point, the point is *annotated* with the postcondition. Given a point (R, i) and a variable assignment v, first we consider the formula 2.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$. We evaluate this as follows:

[1] When $\mathrm{by}_{\mathrm{Id}}(\varphi)$ appears in the precondition of a law, φ need not be the negation of the postcondition. An example is statement (4) in Section 2, which can be represented as:
4.**o**: $\mathrm{by}_{\{1\}}(\Diamond test(z)) \rightsquigarrow \Diamond scr(z)$, where $scr(z)$ is true iff z is tested using a screening test.

- If $(R, i, v) \models d(y) \wedge sp(y)$, (R, i) is *annotated* with 2: $\neg\Diamond test(v(y))$. Observe that the annotation happens regardless of whether $(R, i, v) \models \neg\Diamond test(y)$ and the variable is replaced with the object assigned to it.
- Otherwise, there is no annotation.

Given a variable assignment v and a PredLTL formula φ, $v(\varphi)$ is the formula obtained by replacing all variables x by an identifier for the object $v(x)$. Note that $v(\varphi)$ is equivalent to a propositional LTL formula, as the variables have been replaced by constant symbols. We now define annotations:

Definition 6 (Annotation). *Given a run R, a set of identifiers ID, a variable assignment v, and a body of regulation Reg, an annotation is a statement id: $v(\psi)$ such that $id \in ID$ and id.***x***: $\varphi \rightsquigarrow \psi \in Reg$ (which is either an obligation or a permission). The set of annotations is denoted by $A(R, ID, Reg)$, abbreviated A.*

Definition 7 (Annotation Function). *Given a run R, an annotation function $\alpha : N \rightarrow 2^A$ assigns a set of annotations to each point. Given a set of identifiers ID and $Id \subseteq ID$, we use $\alpha.Id(i)$ to denote the set of annotations id: $\psi \in \alpha(i)$ such that $id \in Id$.*

Time	Objects	Predicates	Annotations
1	o_1	$d(o_1)$, $sp(o_1)$, $\neg test(o_1)$	2: $\neg\Diamond test(o_1)$
2	o_1	$d(o_1)$, $sp(o_1)$, $\neg test(o_1)$	2: $\neg\Diamond test(o_1)$
	o_2	$d(o_2)$, $\neg sp(o_2)$, $\neg test(o_2)$	1: $\Diamond test(o_2)$
3	o_1	$d(o_1)$, $sp(o_1)$, $test(o_1)$	2: $\neg\Diamond test(o_1)$
	o_2	$d(o_2)$, $\neg sp(o_2)$, $\neg test(o_2)$	1: $\Diamond test(o_2)$

Table 2. A run and its annotations

Table 2 shows a run of a bloodbank augmented with annotations. As we discussed in Section 3.1, o_1 is a donation of source plasma which is tested at time 3 and o_2 is a non-source plasma donation which has not been tested. Unless the run is extended to test o_2 as well, it does not conform with the regulation according to Definition 5.

Since the precondition of statement (2) is true for the assignment of y to o_1, we have the annotation 2: $\neg\Diamond test(o_1)$ at all points. However, since o_2 is not a donation of source plasma, there is no correponding annotation.

Now consider the formula $\mathrm{by}_{\{2\}}(\neg\Diamond test(x))$. This is evaluated as follows:

- Evaluate 2.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$ at (R, i) w.r.t. all variable assignments.
- Let ψ_2 be the conjunction of the annotations produced by the formula for (2), i.e., $\psi_2 = \bigwedge \varphi$ for all $\varphi \in \alpha.\{2\}(i)$.
- $(R, i, v) \models \mathrm{by}_{\{2\}}(\neg\Diamond test(x))$ iff $\models \psi_2 \Rightarrow \neg\Diamond test(v(x))$.

Notice that the last step requires a validity check, but it is a validity check in (propositional) LTL. Validity in LTL is coNP-complete when the only modality is *globally*, and

PSPACE-complete with the *until* modality [12]. In Section 4, we discuss cases where the size of the validity tests grows large, and we explore a restriction that lets us avoid validity tests during checking.

Returning to the run in Table 2, the states are annotated with 2: $\neg\Diamond test(o_1)$ and $\models \neg\Diamond test(o_1) \Rightarrow \neg\Diamond test(o_1)$, since $\varphi \Rightarrow \varphi$ is a propositional tautology. So $(R, i, v) \models \mathrm{by}_{\{2\}}(\neg\Diamond test(x))$ when $v(x) = o_1$.

We can evaluate 1.**o**: $d(x) \wedge \neg\mathrm{by}_{\{2\}}(\neg\Diamond test(x)) \rightsquigarrow \Diamond test(x)$ similarly by annotating states with $\Diamond test(x)$ if the precondition holds. In Table 2, this results in an annotation of 1: $\Diamond test(o_2)$ on the appropriate states. If o_2 is never tested, the run will be declared non-conforming (by Definition 5), but the annotation will remain. This lets a law which depends on (1) draw the correct inference.

The semantic evaluation outlined above works only when the references are acyclic, since an order of evaluation needs to be defined. To handle cycles, we move to a three-valued logic where the third (middle) value stands for undetermined. Initially, all statements are undetermined, and there are no annotations. At each step we assign truth values and annotations, using truth values and annotations from the previous step, until we reach a fixed point. In a companion paper [10], which focusses on the design of the logic, we prove that there is a least fixed point, which can be computed in an iterative fashion. In this paper, we use the existence of the least fixed point to derive a runtime checking algorithm.

4 Runtime Checking of Specifications with References

4.1 An Algorithm for Evaluating Specifications with References

We augment the evaluation procedure of the rule-based formalism Eagle [5] to handle references. In Eagle, formulas in LTL are evaluated by transforming them into other formulas, and discharging the remainder (if any) at trace end. The update calculus used in [5] provides a general treatment of past modalities and data dependencies. To simplify presentation, we will work directly with the formulas in the logic.

The key idea is to treat the predicate $\mathrm{by}_{\mathrm{Id}}(\varphi)$ as kind of eventuality. As we discussed in Section 3.3, to evaluate $\mathrm{by}_{\mathrm{Id}}(\varphi)$ at time i, we need to check the annotations obtained from the laws in Id at time i. If the preconditions of the laws in Id are temporal, we need to wait until they are evaluated before the annotations are obtained. So, we need to keep annotations for a time i until all subformulas $\mathrm{by}_{\mathrm{Id}}(\varphi)$ for time i have been evaluated. Given $\mathrm{by}_{\mathrm{Id}}(\varphi)$ and a time i, we attempt to evaluate it using the current set of annotations. If we cannot determine the truth value, $\mathrm{by}_{\mathrm{Id}}(\varphi)$ is transformed into $\mathrm{by}_{\mathrm{Id}}(\varphi, i)$ (read as "$\mathrm{by}_{\mathrm{Id}}(\varphi)$ is true at time i"), and evaluated at subsequent times.

Following [13], we use a three-valued logic with values from $\mathcal{B}^3 = \{\top, \bot, ?\}$, with the meaning true, false, and undetermined, respectively. For notational simplicity, we use truth values as terms in preconditions:

Definition 8 (Syntax of Preconditions). *Given sets $\Phi_1, ..., \Phi_n$ (of predicate names), a set of variables X, and a finite set of identifiers ID, the language $L'(\Phi_1, ..., \Phi_n, X, ID)$, abbreviated as L', is the smallest set such that:*

– If $t \in \mathcal{B}^3$, then $t \in L'$. $p(y_1, ..., y_j) \in L'$, where $p \in \Phi_j$ and $(y_1, ..., y_j) \in X^j$.

– If $\varphi \in L'$, then $\neg\varphi \in L'$ and $\Box\varphi \in L'$. If $\varphi, \psi \in L'$, then $\varphi \wedge \psi \in L'$
– If $Id \subseteq ID$ and $\varphi \in L(\Phi_1, ..., \Phi_n, X)$ (Definition 2), then $\mathrm{by}_{\mathrm{Id}}(\varphi) \in L'$. In addition, for all natural numbers $i \in N$, $\mathrm{by}_{\mathrm{Id}}(\varphi, i) \in L'$

The syntax of regulatory statements (Definition 3) is modified so that the preconditions of laws are statements from L'. The set L' together with a set of regulatory statements Reg is denoted by $L^+ = L' \cup Reg$. Given a set of objects O, $V(X, O)$ denotes the set of all variable assigments, i.e., functions $v : X \to O$.

We can now adapt the Eagle procedure of transforming formulas. The transformation function uses two annotation functions α and α' such that for all i, $\alpha(i) \subseteq \alpha'(i)$. $\alpha(i)$ is the set of annotations obtained from laws with true preconditions, while $\alpha'(i)$ is set of annotations from laws with true or undetermined preconditions. The truth of $\mathrm{by}_{\mathrm{Id}}(\varphi)$ is determined using α, and falsity is determined using α'.

Definition 9 (Transformation function). *Given a set of objects O and annotation functions α and α' such that $\alpha(i) \subseteq \alpha'(i)$ for all $i \in N$, the transformation function $\tau_{(\alpha,\alpha')} : L^+ \times S \times N \times V(X, O) \to L^+$ is defined as follows:*

- $\tau_{(\alpha,\alpha')}(t, s, i, v) = t$ *if* $t \in \mathcal{B}^3$.
- $\tau_{(\alpha,\alpha')}(p(y_1, ..., y_j), s, i, v) = \top$ *if* $(v(y_1), ..., v(y_j)) \in \pi_j(p, s)$.
  $\tau_{(\alpha,\alpha')}(p(y_1, ..., y_j), s, i, v) = \bot$ *otherwise.*
- $\tau_{(\alpha,\alpha')}(\varphi \wedge \psi, s, i, v) = \tau_{(\alpha,\alpha')}(\varphi, s, i, v) \wedge \tau_{(\alpha,\alpha')}(\psi, s, i, v)$.
  $\tau_{(\alpha,\alpha')}(\neg\varphi, s, i, v) = \neg\tau_{(\alpha,\alpha')}(\varphi, s, i, v)$
- $\tau_{(\alpha,\alpha')}(\Box\varphi, s, i, v) = \tau_{(\alpha,\alpha')}(\varphi, s, i, v) \wedge \Box\varphi$
- $\tau_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi), s, i, v) = \tau_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, i), s, i, v)$

$$\tau_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, j), s, i, v) = \begin{cases} \top \text{ if } j \leq i \text{ and } \bigwedge \alpha.Id(j) \wedge v(\neg\varphi) \text{ is not satisfiable} \\ \bot \text{ if } j \leq i \text{ and } \bigwedge \alpha'.Id(j) \wedge v(\neg\varphi) \text{ is satisfiable} \\ \mathrm{by}_{\mathrm{Id}}(\varphi, j) \text{ otherwise} \end{cases}$$

- $\tau_{(\alpha,\alpha')}(id.\mathbf{o}: \varphi \rightsquigarrow \psi, s, i, v) = id.\mathbf{o}: \tau_{(\alpha,\alpha')}(\varphi, s, i, v) \rightsquigarrow \tau_{(\alpha,\alpha')}(\psi, s, i, v)$
  $\tau_{(\alpha,\alpha')}(id.\mathbf{p}: \varphi \rightsquigarrow \psi, s, i, v) = id.\mathbf{p}: \tau_{(\alpha,\alpha')}(\varphi, s, i, v) \rightsquigarrow \psi$

Note that the postcondition of permissions are not transformed, as their truth value is irrelevant. The only use of postconditions of permissions is to provide annotations. To update the annotation function, we need to know if a precondition has become true or false. We now define a function to map formulas to truth values:

Definition 10. *Given a set of objects O and annotation functions α and α' such that $\alpha(i) \subseteq \alpha'(i)$ for all $i \in N$, the function $\mathbf{value}_{(\alpha,\alpha')} : L^+ \times S \times N \times V(X, O) \to \mathcal{B}^3$ is defined as follows:*

- *Truth values, predicates, conjunction and negation are handled in the usual way.*
- $\mathbf{value}_{(\alpha,\alpha')}(\Box\varphi, s, i, v) = \top$ *if s is the final state.*
  $\mathbf{value}_{(\alpha,\alpha')}(\Box\varphi, s, i, v) = ?$ *otherwise.*
- $\mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi), s, i, v) = \mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, i), s, i, v)$

$$\mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, j), s, i, v) = \begin{cases} \top \text{ if } j \leq i \text{ and } \bigwedge \alpha.Id(j) \wedge v(\neg\varphi) \text{ is not satisfiable} \\ \bot \text{ if } j \leq i \text{ and } \bigwedge \alpha'.Id(j) \wedge v(\neg\varphi) \text{ is satisfiable} \\ ? \text{ otherwise} \end{cases}$$

– $\mathbf{value}_{(\alpha,\alpha')}(id.\mathbf{o}\text{: } \varphi \rightsquigarrow \psi, s, i, v) = \mathbf{value}_{(\alpha,\alpha')}(\varphi \Rightarrow \psi, s, i, v)$.
$\mathbf{value}_{(\alpha,\alpha')}(id.\mathbf{p}\text{: } \varphi \rightsquigarrow \psi, s, i, v) = \top$

At the end of the trace, subformulas $\Box\varphi$ are replaced by $\top$, but subformulas $\mathrm{by}_{\mathrm{Id}}(\varphi, j)$ may still be undetermined. This is due to the fact that with circular references, we can create paradoxical statements – id.**o**: $\neg\mathrm{by}_{\{\mathrm{id}\}}(\varphi) \rightsquigarrow \varphi$. This statement requires φ to hold when it doesn't require φ, and is always undetermined.

Update($Reg, \Phi, \alpha, \alpha', s, i$**):**
Input: The regulation Reg, the set of formulas to be updated Φ, the annotation functions α and α', the state s and time i
Let $\alpha(i) = \alpha'(i) = \emptyset$;
for *all id.***x***:* $\varphi \rightsquigarrow \psi \in Reg$ *and assignments* v **do**
  Let $\phi = \tau_{(\alpha,\alpha')}$(id.**x**: $\varphi \rightsquigarrow \psi, s, i, v$);
  $\Phi = \Phi \cup \{(\phi$,id: $v(\psi), i, v)\}$, and $\alpha'(i) = \alpha'(i) \cup \{$id: $v(\psi)\}$
end
repeat
  for *all* (*id.***x***:* $\varphi \rightsquigarrow \psi, a, j, v) \in \Phi$ **do**
    If $\mathbf{value}(\varphi, s, i, v) = \top$, then $\alpha(j) = \alpha(j) \cup \{a\}$;
    If $\mathbf{value}(\varphi, s, i, v) = \bot$, then $\alpha'(j) = \alpha'(j) - \{a\}$
  end
  Let $\Phi' = \emptyset$;
  for *all* (*id.***x***:* $\varphi \rightsquigarrow \psi, a, j, v) \in \Phi$ **do**
    Let $\phi = \tau_{(\alpha,\alpha')}$(id.**x**: $\varphi \rightsquigarrow \psi, s, i, v$) and $\varphi' = \tau_{(\alpha,\alpha')}(\varphi, s, i, v)$;
    If $\mathbf{value}(\phi, s, i, v) =?$ or $\mathbf{value}(\varphi', s, i, v) =?$, then $\Phi' = \Phi' \cup \{(\phi, j)\}$;
    If $\mathbf{value}(\phi, s, i, v) = \bot$, then raise alarm.
  end
  $\Phi = \Phi'$
until α *and* α' *do not change* ;

Algorithm 1: An algorithm for evaluating statements with references

We note that the function $\mathbf{value}_{(\alpha,\alpha')}$ does not determine a formula to be true or false as early as possible. To decide if a formula is true as early as possible, we need to check whether all possible suffixes to the trace satisfy the formula, as in [13]. In other words, we need to decide if the transformed formula is valid. In [10], we show that with references one can encode formulas in first-order logic as regulations, and as a result, the validity problem is undecidable for L^+. The satisfiability tests used to evaluate the inference predicates are in propositional LTL, and are decidable.

Fixed points are defined at the level of a run. Suppose we are given a body of regulation Reg, a run R and annotation functions (α_1, α'_1). The result of evaluation gives us new annotations (α_2, α'_2) corresponding to laws that have true preconditions (α_2), and true or undetermined preconditions (α'_2). We will say that (α_1, α'_1) is a fixed point iff $(\alpha_1, \alpha'_1) = (\alpha_2, \alpha'_2)$.

The function $\mathbf{value}_{(\alpha,\alpha')}$ is extended to runs. The definition remains identical except that for $\mathrm{by}_{\mathrm{Id}}(\varphi, j)$ we do not require that $j \leq i$ to determine truth or falsity, and for the temporal operator:

$$\mathbf{value}_{(\alpha,\alpha')}(\Box\varphi, R, i, v) = \begin{cases} \top \text{ if for all } j \geq i, \mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = \top \\ \bot \text{ if there exists } j \geq i, \mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = \bot \\ ? \text{ otherwise} \end{cases}$$

Definition 11 (Consistent Annotations). *Given a body of regulation Reg and a run R with a set of objects O, the pair of annotation functions (α, α') is consistent iff for all (id.***x***: $\varphi \rightsquigarrow \psi, i, v) \in Reg \times N \times V(X, O)$:*
If id: $v(\psi) \in \alpha(i) \cap \alpha'(i)$, then $\mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = \top$
If id: $v(\psi) \notin \alpha(i) \cup \alpha'(i)$, then $\mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = \bot$
In addition, for all i, we require that $\alpha(i) \subseteq \alpha'(i)$.

Definition 12 (Fixed Point). *Given a body of regulation Reg and a run R with a set of objects O, the pair of consistent annotation functions (α, α') is a fixed point iff for all (id.***x***: $\varphi \rightsquigarrow \psi, i, v) \in Reg \times N \times V(X, O)$:*
If $\mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = \top$, then id: $v(\psi) \in \alpha(i) \cap \alpha'(i)$
If $\mathbf{value}_{(\alpha,\alpha')}(\varphi, R, i, v) = ?$, then id: $v(\psi) \in \alpha'(i) - \alpha(i)$
Otherwise, id: $v(\psi) \notin \alpha(i) \cup \alpha'(i)$

We say that $(\alpha_1, \alpha'_1) \leq (\alpha_2, \alpha'_2)$ if for all i, we have $\alpha_1(i) \subseteq \alpha_2(i)$. We now review some results that are proved in [10]. The partially ordered set of consistent annotations has a least fixed point and one or more maximal fixed points. Distinct fixed points arise if there are circular references. The converse is not necessarily true, i.e., there may be circular references and a unique fixed point. There is a smallest element in the set of consistent annotations (α_0, α'_0) such that for all i, $\alpha_0(i) = \emptyset$ and $\alpha'_0(i)$ contains all annotations. The least fixed point can be obtained iteratively using (α_0, α'_0).

Algorithm 1 describes the procedure for computing the least fixed point in a runtime setting. In addition to α and α', we maintain a set of tuples Φ, where each element is a transformed regulatory statement, the associated annotation, time and variable assignment. Given (id.**x**: $\varphi \rightsquigarrow \psi, a, j, v) \in \Phi$, if φ is determined to be true, the annotation a is added to $\alpha(j)$. On the other hand, if φ is determined to be false a is removed from $\alpha'(j)$. For all $j \in N$, $\alpha(j)$ increases monotonically, and $\alpha'(j)$ decreases monotonically with each execution of the repeat loop, until a fixed point is reached.

4.2 Complexity Analysis by Example

The complexity of Algorithm 1 in each state of a run depends on two factors – the number of steps necessary to reach a fixed point, and the size of satisfiability problem instances that need to be handled in the evaluation of the predicate $\mathrm{by}_{\mathrm{Id}}(\varphi)$. We discuss examples that illustrate these two aspects, by encoding the graph reachability problem in different ways. In the first example, the number of steps taken to reach the fixed point grows with the number of objects. In the second example, the size of the satisfiability instances grows with the number of objects.

Both examples operate on the same model, where a state in the run contains a description of a graph. Objects o_1 and o_2 represent nodes, and the predicate $\delta(o_1, o_2)$ is

true iff there is an edge between o_1 and o_2. In addition, $\delta^+(o_1, o_2)$ is true iff there is a path from o_1 to o_2. Suppose we wish to check whether δ^+ has been computed correctly.

Example 1. Consider a self-referential sentence:

id.**o**: $\delta(x,z) \vee (\delta(x,y) \wedge \mathrm{by}_{\{\mathrm{id}\}}(\delta^+(y,z))) \rightsquigarrow \delta^+(x,z)$

The precondition of this sentence corresponds to the definition of a path. In other words, there is a path between x and z ($\delta^+(x,z)$), if there is an edge between x and y ($\delta(x,y)$), and a path between y and z ($\mathrm{by}_{\{\mathrm{id}\}}(\delta^+(y,z))$). Consider the sequence of annotations obtained in the least fixed point computation – $\alpha_0, ..., \alpha_f$. It is easy to see that id: $\delta^+(o,o') \in \alpha_j(i)$ iff there is a path of length at most j from o to o'. Given a graph with $|O|$ nodes, there is a path from o to o' iff there is a path of length at most $|O|$ from o to o'. As a result, the fixed point will be reached in at most $|O|$ steps. The worst-case number of steps needed to reach the fixed point is $\mathbf{O}(m \times |O|^k)$, where m is the size of the regulation, and k is the maximum number of variables appearing in a sentence.

Example 2. Consider now the following statements:

A.**o**: $\mathrm{by}_{\{\mathrm{B,C}\}}(\delta^+(x,y)) \rightsquigarrow \delta^+(x,y)$
B.**o**: $\delta(x,y) \rightsquigarrow \delta^+(x,y)$
C.**o**: $\top \rightsquigarrow (\delta^+(x,y) \wedge \delta^+(y,z)) \Rightarrow \delta^+(x,z)$

Note that A refers to C. The presence of implication in the postcondition of C is an important feature of this example. Let, for simplicity, the graph in the state be a chain. Since the precondition of C is always true, the first step of the fixed point computation yields an annotation that contains C: $(\delta^+(o,o') \wedge \delta^+(o',o'')) \Rightarrow \delta^+(o,o'') \in \alpha_1(i)$ for all o, o', o'' in the graph. The next step of the evaluation will yield the fixed point, but the size of the validity test performed in this step is $\mathbf{O}(|O|^3)$, as Algorithm 1 uses all the available annotations. The worst-case size of the validity instances is in $\mathbf{O}(m \times |O|^k)$, and the time complexity of a step in computing the fixed point is $\mathbf{O}(2^{m \times |O|^k})$.

Discussion. In both examples above, Algorithm 1 checks validity instances of size polynomial in $|O|$. However, there is a crucial difference in the maximum size of tests that are needed. In Example 1, $\mathrm{by}_{\{\mathrm{id}\}}(\delta^+(o,o'))$ is true iff id: $\delta^+(o,o') \in \alpha(i)$. In other words, at most one annotation is need to evaluate $\mathrm{by}_{\{\mathrm{id}\}}(\delta^+(o,o'))$. In Example 2, we do need validity tests of size $|O|$ to evaluate $\mathrm{by}_{\{\mathrm{B,C}\}}(\delta^+(o,o'))$. A case study of the CFR revealed that the references behaved like Example 1 in that a single annotation or copy of the referenced statement suffices to evaluate formulas $\mathrm{by}_{\mathrm{Id}}(\varphi)$. We call this *the single copy property*.

Definition 13 (Single Copy Property). *Given a body of regulation Reg, $\mathrm{by}_{\mathrm{Id}}(\varphi, j)$ has the single copy property iff for all runs R, and consistent annotations (α, α'):*

$$t = \begin{cases} \top \textit{ if } \psi \wedge v(\neg\varphi) \textit{ is not satisfiable for some } \psi \in \alpha.Id(j) \\ \bot \textit{ if } \psi \wedge v(\neg\varphi) \textit{ is satisfiable for all } \psi \in \alpha'.Id(j) \\ ? \textit{ otherwise} \end{cases}$$

where, $t = \mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, j), s, i, v)$

While the single copy property allows us to reduce the size of the satisfiability tests, we need to perform $\mathbf{O}(m \times |O|^k)$ tests for each inference predicate. The question arises as to whether satisfiability tests can be avoided during checking. We answer this question positively in the following section.

4.3 Pre-computing Satisfiability

Algorithm 1 evaluates $\mathrm{by}_{\mathrm{Id}}(\varphi)$ using satsfiabilty tests. The size of the satisfiability tests depends on $\alpha.Id(i)$, which in turn depends on the number of objects. If $\mathrm{by}_{\mathrm{Id}}(\varphi)$ has the single copy property, we can consider smaller satisfiability tests. In this section, we show that the single copy property gives us a way to assess satisfiability symbolically, and use tests of lower complexity during checking.

The strategy we use is as follows. Given a body of regulation, we perform a compilation step which involves: a) testing satisfiability, and b) replacing the predicates $\mathrm{by}_{\mathrm{Id}}(\varphi)$ by equivalent formulas in another logic. We begin by discussing two examples, and then formalize the compilation step.

Example 1: Consider our regulatory sentences:

- 1.**o**: $d(x) \wedge \neg \mathrm{by}_{\{2\}}(\neg \Diamond test(x)) \rightsquigarrow \Diamond test(x)$, and
- 2.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg \Diamond test(y)$

Consider a state at which o_1, o_2, ..., o_n are source plasma donations. This would result in $\neg \Diamond test(o_1)$, $\neg \Diamond test(o_2)$, ..., $\neg \Diamond test(o_n)$ being available as annotations. To evaluate $\mathrm{by}_{\mathrm{Id}}(\neg \Diamond test(o_i))$, Algorithm 1 uses all the annotations in the satifiability test. However, in this case, it suffices to check if $\neg \Diamond test(o_i)$ is present as an annotation. The other annotations are irrelevant. To check if $\neg \Diamond test(o_i)$ is present as an annotation, it suffices to evaluate the precondition of the referenced law, i.e., whether $d(o_i) \wedge sp(o_i)$ is true (whether o_i is a donation of source plasma). Instead of evaluating $\mathrm{by}_{\mathrm{Id}}(\phi)$ using satisfiability tests, we will check if the precondition of a referenced law is true.

Informally, the compilation step involves anwering the question *when does statement 2 provide an exception for statement 1*. Equivalently, when does $\neg \Diamond test(y)$ imply $\neg \Diamond test(x)$. The answer is only when $y = x$. We can then evaluate the precondition of 2 with y replaced by x, i.e., $d(x) \wedge sp(x)$. This lets us replace statement 1 with 1.**o**: $d(x) \wedge \neg(d(x) \wedge sp(x)) \rightsquigarrow \Diamond test(x)$, which is equivalent to 1.**o**: $d(x) \wedge \neg sp(x) \rightsquigarrow \Diamond test(x)$. Observe that this is the derived obligation implied by statements 1 and 2, i.e., every non-source plasma donation must be tested.

Example 2: The example above is simple in two ways: a) the number of variables in both statements are the same, and b) the references are acyclic. We discuss the general case in the context of the reachability example we saw in the previous section:

id.**o**: $\delta(x, z) \vee (\delta(x, y) \wedge \mathrm{by}_{\{\mathrm{id}\}}(\delta^+(y, z))) \rightsquigarrow \delta^+(x, z)$

We observe that the precondition is structurally similar to a procedure that checks if a path exists between two nodes x and z. That is, if $\delta(x, z)$ then $\delta^+(x, z)$ is true. Otherwise, if there exists y such that $\delta(x, y)$ and there is a path from y to z, then $\delta^+(x, z)$ is true, otherwise false.

We will produce a formula which mimics the procedure. There are two pieces of machinery used by the procedure that are not directly available in the logic: a) an existential quantifier over objects (there exists y), and b) a mechanism for recursion. To address this, let us consider a logic which extends PredLTL with existential quantifiers, and a function symbol P_{id} for $id \in ID$ (P stands for precondition). P_{id} takes as argument a substitution $\theta : X \rightarrow X$, which is a function from variables to variables. A substitution is represented a set of replacements x/y (read as "x is replaced by y"), such that each variable has at most one replacement. We replace the formula above with:

id.**o**: $\delta(x,z) \vee (\delta(x,y) \wedge \exists y_1 : \mathrm{P}_{\mathrm{id}}(\{x/y, y/y_1, z/z\})) \rightsquigarrow \delta^+(x,z)$

It remains to give this formula a semantics. Given a variable assignment v and a subtitution θ, $\theta(v)$ denotes the variable assignment v' such that $v'(x) = v(\theta(y))$. Given a run R, time i and regulation Reg, the idea is to say that $(R,i,v) \models \mathrm{P}_{\mathrm{id}}(\theta)$ iff $(R,i,\theta(v)) \models \varphi$ where id.**x**: $\varphi \rightsquigarrow \psi \in Reg$. We now formalize the compilation procedure.

Compiling References into Precondition Tests: We begin by defining the syntax of compiled preconditions:

Definition 14 (Syntax of Compiled Preconditions). *Given sets $\Phi_1, ..., \Phi_n$ (of predicate names), a set of variables X, and a finite set of identifiers ID, the language $L'_C(\Phi_1, ..., \Phi_n, X, ID)$, abbreviated as L'_C, is the smallest set such that:*

- *If $t \in \mathcal{B}^3$, $t \in L'_C$. And, $p(y_1, ..., y_j) \in L'_C$ where $p \in \Phi_j$ and $(y_1, ..., y_j) \in X^j$.*
- *If $\varphi \in L'_C$, then $\neg\varphi \in L'_C$ and $\Box\varphi \in L'_C$. If $\varphi, \psi \in L'_C$, then $\varphi \wedge \psi \in L'_C$*
- *If $\varphi \in L'_C$, for all $y \in X$, we have $\exists y : \varphi \in L'_C$.*
- *For all $id \in Id$ and substitutions $\theta : X \rightarrow X$, we have $\mathrm{P}_{\mathrm{id}}(\theta) \in L'_C$. In addition, for all natural numbers $i \in N$, $\mathrm{P}_{\mathrm{id}}(\theta, i) \in L'_C$*

The syntax of regulatory statements (Definition 3) is modified so that the preconditions of laws are statements from L'_C. The set L'_C together with a set of regulatory statements Reg_C is denoted by $L^+_C = L'_C \cup Reg_C$. We remind the reader that L^+ and L' are the languages with the predicate $\mathrm{by}_{\mathrm{Id}}(\varphi)$.

The semantics of L^+_C is defined in a manner similar to L^+. Rather than using annotations (α, α'), we now evaluate statements w.r.t. two sets of assignment functions (γ, γ'). $\gamma(i, id)$ (resply., $\gamma'(i, id)$) is a set of variable assignments for which the precondition of the law with identifier id is true (resply., true or undetermined). As with annotations, we require that for all $i \in N$ and $id \in ID$, $\gamma(i, id) \subseteq \gamma'(i, id)$. Given an assignment v and a substitution θ, $\theta(v)$ denotes the assignment v' such that for all $y \in X$, we have $v'(y) = v(\theta(y))$. We can now adapt the $\mathbf{value}$ function:

$$\mathbf{value}_{(\gamma,\gamma')}(\mathrm{P}_{\mathrm{id}}(\theta, j), R, i, v) = \begin{cases} \top \text{ if } \theta(v) \in \gamma(j, id) \\ \bot \text{ if } \theta(v) \notin \gamma'(j, id) \\ ? \text{ otherwise} \end{cases}$$

The definitions of consistency and fixed points (Definitions 11 and 12) are easily adapted, and we leave the details to the reader.

We now describe the compilation procedure. Given $\varphi \in L^+$, we use $X(\varphi)$ to denote the set of variables appearing in φ, and $\theta(\varphi)$ to denote the formula obtained by performing the substitution θ on φ. Consider $\mathrm{by}_{\mathrm{Id}}(\varphi, j)$, which has the single copy property, and variables disjoint from all regulatory statements:

- Let $\mathcal{S}(\varphi, id)$ = { θ| id.**x**: $\varphi \rightsquigarrow \psi \in Reg$, and $\theta(\psi \Rightarrow \varphi)$ is valid}.
- For all $\theta \in \mathcal{S}(\varphi, id)$, let $\varphi_C(\theta, id) = \exists z_1, ..., z_m : \mathrm{P}_{\mathrm{id}}(\theta, j)$, where the existentially quantified variables are in one-to-one correspondence with those in $X(\phi) - X(\psi)$. More formally, $z_j \notin X(\phi) - X(\psi)$ and θ is a one-to-one function from $\{z_j | 1 \leq j \leq m\}$ to $X(\phi) - X(\psi)$.

- $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi, j), id) = \bigvee\{\varphi_C(\theta, id) | \theta \in \mathcal{S}(\varphi, id)\}$, and
- $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi, j)) = \bigvee\{\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi, j), id) | id \in Id\}$

We note that the first step makes crucial use of the single copy property (SCP). In computing $\mathcal{S}(\varphi, id)$, it suffices to find substitutions such that $\theta(\psi \Rightarrow \varphi)$ is valid. If the SCP does not hold, then we need to check if multiple copies of postconditions provide the necessary implication (as in Example 2, Section 4.2). For example, we need to check if $\theta(\psi_1 \wedge ... \wedge \psi_n \Rightarrow \varphi)$, where $\psi_1, ..., \psi_n$ are copies of the postcondition of a law with the variables renamed. It can be shown that detecting whether the SCP holds is undecidable. In future work, we plan to investigate restrictions on postconditions that make SCP-detection decidable.

To prove the correctness of the compilation procedure, we use a notion of correspondence between annotations and assignments. Let us assume as given a body of regulation Reg (in L^+), a run R and consistent annotations (α, α'). Rather than producing a regulation in L^+_C, we prove correctness by evaluating formulas in L'_C against Reg. We construct $(\gamma_\alpha, \gamma'_{\alpha'})$ such that for all $i \in N$ and $id \in ID$, $v \in \gamma_\alpha(i, id)$ iff id: $v(\psi) \in \alpha(i)$, and $v \in \gamma'_{\alpha'}(i, id)$ iff id: $v(\psi) \in \alpha'(i)$. We can now show the following:

Lemma 1. *Given a body of regulation Reg, a run R, consistent annotations (α, α'), and $\mathrm{by}_{\mathrm{Id}}(\varphi, j)$ which has the single copy property, for all $i \in N$ and assignments v:*

$$\mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, j), R, i, v) = \mathbf{value}_{(\gamma_\alpha,\gamma'_{\alpha'})}(\varphi_C(\mathrm{by}_{\mathrm{Id}}(\theta, j)), R, i, v)$$

Proof. The proof follows straightforwardly from the construction of $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\theta, j))$ and the single copy property. We sketch one of the cases.

Suppose $\mathbf{value}_{(\alpha,\alpha')}(\mathrm{by}_{\mathrm{Id}}(\varphi, j), R, i, v) = \top$. There exists id: $v'(\psi) \in \alpha(i)$ such that $v'(\psi) \wedge v(\neg\varphi)$ is not satisfiable, or equivalently $v'(\psi) \Rightarrow v(\varphi)$ is valid. It follows that there exists a substitution θ such that $\theta(\psi \Rightarrow \varphi)$ is valid. By definition $v' \in \gamma_\alpha(i)$, and hence, $\mathbf{value}_{(\gamma_\alpha,\gamma'_{\alpha'})}(\mathrm{P}_{\mathrm{id}}(\theta, j), R, i, v') = \top$. We can then argue using the construction that $\mathbf{value}_{(\gamma_\alpha,\gamma'_{\alpha'})}(\exists z_1, ..., z_m : \mathrm{P}_{\mathrm{id}}(\theta, j), R, i, v) = \top$, and as a result, $\mathbf{value}_{(\gamma_\alpha,\gamma'_{\alpha'})}(\varphi_C(\mathrm{by}_{\mathrm{Id}}(\theta, j)), R, i, v) = \top$. The other cases are handled similarly. □

Given Reg in which all subformulas $\mathrm{by}_{\mathrm{Id}}(\varphi)$ have the single copy property, we can now produce the regulation Reg_C in L^+_C with all occurences of $\mathrm{by}_{\mathrm{Id}}(\varphi)$ replaced by $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi))$. It follows from Lemma 1 that if (α, α') is a fixed point w.r.t. Reg, then $(\gamma_\alpha, \gamma'_{\alpha'})$ is a fixed point w.r.t. Reg_C. In addition, the truth values assigned to regulatory statements are identical.

The complexity of evaluation depends on the number of disjuncts in $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi))$, which in turn depends on the size of the set: $\mathcal{S}(\varphi, id)$. $|\mathcal{S}(\varphi, id)| \leq (2k)^{2k}$, where k is the maximum number of variables in a regulatory statement. $(2k)^{2k}$ is a bound on the number of equivalence classes, i.e., we have $2k$ variables (k in φ and k in ψ) and at most one equivalence class for each variable. Hence, the size of $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi))$ is $\mathbf{O}(m \times (2k)^{2k})$, where m is the number of regulatory statements. Each quantified precondition test can be evaluated in $\mathbf{O}(|O|^k)$ time, where O is the set of objects. As a result, the time complexity for evaluating $\varphi_C(\mathrm{by}_{\mathrm{Id}}(\varphi))$ is $\mathbf{O}(m \times (2k)^{2k} \times |O|^k)$. We now describe an evaluation of the system, using a prototype implementation which performs this compilation procedure.

4.4 Evaluation

We have developed a prototype implementation of the checker. We briefly describe two aspects of the implementation: (a) the interface between regulations and traces (schemas), and (b) the trace-checker.

Schemas form the interface between the regulation and trace. A schema is a set of class and type definitions. Classes can inherit from others, and have attributes which have atomic types, tuples or unions of types, pointers to other objects or sets of values.

Our current implementation of the trace-checker is static in the sense that the entire trace is stored on disk (in an NDBM database). The objects at each state belong to classes in a given schema. The regulation, which is type-checked against the same schema, is compiled using the techniques discussed in Section 4.3, and evaluated at each state. We do not have any special optimizations for speed. The objects are stored as strings, and reparsed every time they are loaded into memory. The checker evaluates each obligation w.r.t. all variable assignments, loading into memory a single variable assignment at a time.

We now describe a preliminary evaluation of the implementation. Our goal was to check if we could scale to traces with a large number of objects, rather than very long traces. We created a schema based on the CFR, capturing donors, donations of several types, and various tests. We then checked a number of synthetic (final) states for conformance. Given a schema, we generate a set of donors by choosing random values for atomic attributes. For each donor we generate a set of donations again choosing attribute values at random. Each donation is randomly tested as follows: with $p = 0.3$ it is tested for all diseases with negative results, with $p = 0.3$ it is test for diseases with a random result, and otherwise it is not tested.

On the regulatory side, we created logic formulas for a portion of the CFR 610.40. A total of 12 sentences, and a list of 6 disease names were used. Lists are frequent in regulation, and statements refer to particular list items. Of the 12 sentences, 7 were obligations and 5 were permissions. A total of 8 reference formulas ($\mathrm{by}_{\mathrm{Id}}(\varphi)$) were used, and of these 3 referred to list items. The compilation step of removing the references took 26 seconds with a total of 96 satisfiability tests. Each statement had at most 2 variables (one for donations and the other for disease names).

We evaluated performance of the checker against a number of states. The number of disease names was 8, and the number of donations varied. The time taken varied linearly with a number of donations. For states with 100, 1000, 5000, and 10000 donations the conformance check took 12s, 130s, 500s, and 1042s respectively. The performance suggests that the approach is practical for checking short traces. However, more incremental algorithms are needed to deploy such specifications in a runtime setting.

5 Discussion and Conclusions

We have described a logic for representing regulatory documents for the application of conformance checking. The logic allows statements to refer to others for conditions or exceptions. While references give us a way to represent regulation directly, the evaluation of references during checking has high complexity. Algorithm 1 uses satisfiability tests of size polynomial in the number of objects. In Sections 4.2 and 4.3, we

described an emprically motivated assumption (*the single copy property*), which lets us replace satifiablity tests with tests of lower complexity. The evaluation of our prototype implementation suggests that the approach is suitable for conformance audits of medium-sized traces.

An important part of making this approach useful in practice is to provide support for translating the regulatory documents into their formal representation. Such support has to rely heavily on natural language processing techniques, which require substantial extension of current state of the art. We are actively pursuing this line of research. Preliminary results are reported in [14, 15].

Related Work. The use of logic to represent and reason about regulation has been of interest for several years. We begin by discussing the literature in relation to two issues: a) the representation of obligation and permission, and b) references between laws. We compare our work with other approaches to conformance checking, and place it in the context of previous work on run-time checking of LTL.

The goal of deontic logic is a formalization of concepts such as obligation, permission and rights. There are many systems of deontic logic, but the most common approach is to treat obligation and permission as modal operators [16, 17]. The logic developed here focusses on the problem of references between laws, and we believe that the representation of obligation and permission is an important but orthogonal issue. In future work, we plan to add a modal treatment of obligation and permission to our system.

The problem of references between laws has been observed in regulatory texts in different domains [18, 2]. More generally, the function of sentences as conditions or exceptions to others has been studied in a variety of contexts. Alchourron and Makinson [19] proposed a hierarchical structure for a legal discourse, to handle exceptions to statements. This led to the development of input-output logic [20], which is closely related to default logic [11]. Previous work on applying default logic has been mainly in the context of computing extensions to a theory, in the manner of logic programs [7, 18, 6]. We believe that the application of these ideas in conformance checking is novel.

Conformance checking has been receiving increasing attention in recent years [1–3, 21]. [1] represents business contracts as SQL queries. [3, 21] use a logic on a UML description of a domain. While the approaches of [1, 3, 21] are similar in spirit to ours, they do not provide a treatment of references. [2] discusses the problem of references in the context of privacy regulation, and the references are resolved manually.

Our work builds upon the well-established work on run-time checking of LTL and its extensions. We have adapted the calculus of Eagle [5] to handle references. Rule-based formalisms [5, 22] are quite general, but the transformation of formulas at each state can be expensive. Automata-based approaches [13] offer a more efficient alternative at the price of generality. We are currently exploring ways to adapt the automata-based approach to our setting.

References

1. Abrahams, A.: Developing and Executing Electronic Commerce Applications with Occurrences. PhD thesis, Univeristy of Cambridge (2002)

2. Breaux, T.D., Vail, M.W., Anton, A.I.: Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In: Proceedings of the 14th IEEE International Requirements Engineering Conference. (2006)
3. Giblin, C., Liu, A., Muller, S., Pfitzmann, B., Zhou, X.: Regulations Expressed as Logical Models (REALM). In Moens, M.F., Spyns, P., eds.: Legal Knowledge and Information Systems. (2005)
4. U.S. Food and Drug Administration: Code of Federal Regulations. http://www.gpoaccess.gov/cfr/index.html
5. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-based runtime verification. In: Proceedings of 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2004). Volume 2937 of LNCS. (January 2004) 44–57
6. McCarty, L.T.: A language for legal discourse - i. basic features. In: Proceedings of ICAIL. (1989)
7. Sergot, M., F.Sadri, Kowalski, R., F.Kriwaczek, P.Hammond, Cory, H.: The british nationality act as a logic program. Communications of the ACM **29**(5) (1986) 370–86
8. Ross, A.: Directives and Norms. Routlege and Kegan Paul (1968)
9. Marcus, R.B.: Iterated deontic modalities. Mind **75**(300) (1966)
10. Dinesh, N., Joshi, A., Lee, I., Sokolsky, O.: Reasoning about conditions and exceptions to laws in regulatory conformance checking. In Submission: http://www.cis.upenn.edu/~nikhild/reasoning.pdf (2008)
11. Reiter, R.: A logic for default reasoning. In: Readings in nonmonotonic reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1987) 68–93
12. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logic. ACM **32** (1985) 733–49
13. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06). Volume 4337 of LNCS. (December 2006)
14. Dinesh, N., Joshi, A.K., Lee, I., Webber, B.: Extracting formal specifications from natural language regulatory documents. In: Proceedings of the Fifth International Workshop on Inference in Computational Semantics. (2006)
15. Dinesh, N., Joshi, A., Lee, I., Sokolsky, O.: Logic-based regulatory conformance checking. In: Proceedings of the 14th Monterey Workshop. (2007)
16. von Wright, G.H.: Deontic logic. Mind **60** (1951) 1–15
17. Aqvist, L.: Deontic logic. In Gabbay, D., Guenthner, F., eds.: Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic. (1984) 605–614
18. Bench-Capon, T., Robinson, G., Routen, T., Sergot, M.: Logic programming for large scale applications in law: A formalisation of supplementary benefit legislation. In: Proceedings of the 1st International Conference on AI and Law. (1987)
19. Alchourron, C., Makinson, D.: Hierarchies of regulation and their logic. In Hilpinen, R., ed.: New Studies in Deontic Logic. (1981)
20. Makinson, D., van der Torre, L.: Input/output logics. Journal of Philosophical Logic **29** (2000) 383–408
21. Glasse, E., Engers, T.V., Jacobs, A.: Power: An integrated method for legislation and regulations from their design to their use in e-government services and law enforcement. In Moens, M.F., ed.: Digitale Wetgeving, Digital Legislation. (2003) 175–204
22. Barringer, H., Rydeheard, D., Havelund, K.: Rule systems for run-time monitoring: From Eagle to RuleR. In: Proceedings of the 7^{th} Workshop on Runtime Verification. Volume 4839 of LNCS. (March 2007) 111–125