Extracting Formal Specifications from Natural Language Regulatory Documents

Nikhil Dinesh, Aravind Joshi, and Insup Lee

Department of Computer Science, Univeristy of Pennsylvania,
Philadelphia, PA - 19104 USA
nikhild,joshi,lee@cis.upenn.edu

Bonnie Webber

University of Edinburgh, Edinburgh, EH8 9LW Scotland
bonnie@inf.ed.ac.uk

Abstract
Formal verification techniques provide a way to determine whether regulatory documents are consistent and whether implementations conform to them. To apply these techniques a formal description of the regulation needs to be extracted. We present a framework, under which NLP techniques can be brought to bear, to aid a requirements engineer in extracting the formal description.

1 Introduction

Regulatory documents, which include the vast bodies of legislation, operating procedures and organizational policy, are meant to be accessible to the people affected by them. Hence, they have to be in natural language (NL). On the other hand, regulations are expected to be consistent, and the governed entities/events are expected to conform to the regulation.

For example, the Food and Drug Administration's Code of Federal Regulations (FDA CFR) governs the bloodbanks in America.[1] The bloodbanks perform safety-critical functions like the testing of blood for communicable disease agents (like HIV). It is highly desirable to determine whether (a) the CFR is consistent, and (b) a bloodbank's implementation of such a function conforms to the CFR.

[⋆] This research was supported in part by NSF CCF-0429948 and ARO 911NF-05-1-0158
[1] http://www.gpoaccess.gov/cfr/index.html

The problem of creating descriptions of regulation which can be checked for consistency has been explored by several authors [1,8], but the challenge of checking an implementation for conformance has not been addressed, and this is the main goal of our work. The conformance guarantees can be obtained if formal descriptions of regulation and implementations are available, and if verification techniques [4] can be applied to these descriptions. But extracting a formal description of regulation is expensive, as regulatory bases like the CFR are large (about a million words) and complex.

Formal descriptions of regulation are usually extracted by an individual who has a background in logic, e.g., [1,8]. We will call this individual *the requirements engineer*. In this paper, we describe a framework to assist a requirements engineer in extracting a formal description of regulation for use in conformance checking.

An overview of the framework, the theoretical background and the various constraints that apply is given in Section 2. This lets us determine the nature of the description that needs to be extracted from the regulation. We then turn to the question of how these descriptions might be composed. In Section 3, we attempt to map the denotations of sentences assigned by Kratzer [12] to a form that can be used for the task at hand. Some difficulties arise in this mapping, mainly because notions of *obligation* (that which is required) and *permission* (that which is allowed) are not captured in the denotations. We argue that an account of these notions is essential to the task at hand. Section 4 describes a semantic representation, and composition procedure to assist the requirements engineer in extracting the required description. By treating obligations and permissions as different dimensions of the description computed, the difficulties encountered in Section 3 are addressed.

The approach is motivated by our case study of the FDA CFR, and we use (1) and (2) as examples through the course of this paper.[2] (1) conveys an obligation to perform a test for HIV and Hepatitis B, and (2) conveys a permission not to test source plasma (a blood component) for Hepatitis B.

(1) Except as specified in (2), you must test each donation of human blood or blood component, for evidence of infection due to the Human immunodeficiency virus, and the Hepatitis B virus.

(2) You are not required to test donations of Source Plasma for evidence of infection due to the Hepatitis B virus.

2 A Framework

To determine whether an implementation (bloodbank) conforms to the regulation (CFR), we extract specifications in the Computation Tree Logic (CTL) from the CFR. Then, given a description of a bloodbank's procedure (as a finite transition system, or model) there is an efficient search procedure to

[2] (1) and (2) are modified versions of sentences that appear in the FDA CFR 610.40. The actual sentences are very long, and the modifications are made in the interests of space.

determine if the model conforms to the CTL specification [3]. This is known as temporal model checking [2,13]. The problem of conformance checking is thus split into three steps:

(1) Extract CTL specifications from the regulation - This is done by a requirements engineer, and our goal is to assist her. We use CTL as the specification language, because it allows for efficient model checking [3].

(2) Obtain a model of an implementation - We assume the availability of models. There are tools that aid in extracting models from software [5], and in creating models if they cannot be extracted directly [11].

(3) Apply model checking to determine if the model conforms to the CTL specification.

Formally, a model can be defined as follows:

Definition 2.1 *A model M is the five-tuple (S, I, δ, π, Π), where:*

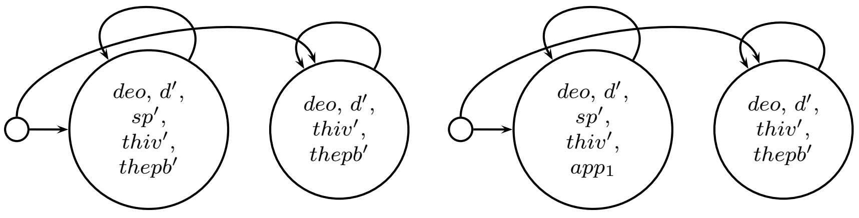
(a) S is a set of states, $I \subseteq S$ is a non-empty set of initial states,

(b) $\delta \subseteq S \times S$ is a total transition relation (that is, $\forall s \in S : [\exists t \in S : (s,t) \in \delta]$),

(c) π is a set of propositions (with power set 2^π), and

(d) $\Pi : S \rightarrow 2^\pi$ is a function from states to sets of propositions. $\Pi(s)$ for $s \in S$ can be thought of as the propositions true at s.

Figure 1(a) and 1(b) show models of two different bloodbanks. The leftmost state is the initial state. Each state is labeled with $\Pi(s)$. The propositions have the following interpretation: d' is true ($d' \in \Pi(s)$) iff a donation of blood or blood component is being processed, sp' is true iff a donation of source plasma is being processed, $thiv'$ is true iff a test for HIV has been performed, and $thepb'$ is true iff a test for Hepatitis B has been performed. The use of the propositions *deo* (denoting *deontic accessibility*) and app_1 (denoting the application of a permission) is explained in later sections.



(a) ***A model of a bloodbank which tests all donations***

(b) ***A model of a bloodbank which does not test donations of source plasma for Hepatitis B***

Fig. 1. Two models of bloodbanks

Definition 2.2 *Given a finite set of propositions π, CTL formulas are defined inductively as follows:*

(a) $p \in \pi$ is a formula,

(b) Boolean combinations and negations of formulas are formulas,

(c) if ϕ, and ψ are formulas, then $AG(\phi)$ (on all paths, globally ϕ), $AX(\phi)$ (on all paths, at the next state ϕ), and $\phi AU \psi)$ (on all paths, ϕ until ψ) are formulas.

The only temporal operator in CTL that we use is AG (for reasons that we describe below), and hence rather than define the interpretation formally, we will give some examples. Let M_1 be the model in Figure 1(a), and M_2 be the model in Figure 1(b). The CTL specification $AG(deo \rightarrow (d' \rightarrow thiv'))$ holds of both models, since on all paths (from the initial state, the leftmost one in Figures 1(a), and 1(b)), globally, in all deontically accessible states deo, if a donation of blood or blood component is being processed d', it is tested for HIV $thiv'$. Hence, we write $M_1 \vDash AG(deo \rightarrow (d' \rightarrow thiv'))$, and $M_2 \vDash AG(deo \rightarrow (d' \rightarrow thiv'))$. Also, $M_1 \vDash AG(deo \rightarrow (sp' \rightarrow thepb'))$. But, $M_2 \nvDash AG(deo \rightarrow (sp' \rightarrow thepb'))$ (since there is a state s with $sp' \in \Pi(s)$, and $thepb' \notin \Pi(s)$).

2.1 Approaches to extracting specifications

The central problem we face is that CTL and other temporal logics that lend themselves to model checking are not expressive enough for a compositional semantic procedure to be defined for natural language. One reason is that CTL, like propositional logic, cannot express relations between entities.

There are several routes one might take to address this problem, i.e., design more expressive logics that allow for tractable model checking, focus on a subset of NL from which an automatic translation is guaranteed, or make the procedure machine-assisted. While the design of more expressive logics makes the composition of specifications easier, using them for model checking needs the creation of more expressive models (which requires more effort). As a result, there is a trade-off between amount of effort spent in obtaining models, and that in obtaining the specifications. Our decision to work with less expressive models is motivated by the extensive tool support available for creating and extracting such models [5,11]. Further, subsets of NL for which automatic translation is guaranteed, such as the one derived by Holt and Klein [10], assume (among other things) that references are resolved and hence cannot be directly applied to regulatory documents. We are thus left with the choice of making the procedure machine-assisted.

There have been two kinds of machine-assisted approaches to extracting temporal logic specifications: (a) composing the semantics in a general semantic framework which is then mapped to temporal logic [7], and (b) attempting to compose the semantics in the temporal logic directly [6]. In the latter approach, a human specifies denotations for a portion of the sentence, and the rest of the composition happens automatically. We attempt to compose the semantics in a temporal logic directly like [6], as it lends itself to defining semantic representations with which a requirements engineer can interact in well-defined ways.

2.2 Constraints on the CTL specifications

We apply two constraints to the CTL specifications:

(i) The specifications extracted should hold of all and only the valid models. There may be several implementations that aim to conform to a single base of regulation. Given (1) and (2), the models in Figures 1(a) and 1(b) are both valid. This is an important difference from the NL sentences considered in previous approaches, which were elicited from appropriate users by presenting them with a single model. For example, Holt and Klein [10] obtained specifications by asking users to describe a particular timing diagram.

(ii) To account for the variation between models, all temporal information about the governed entities/events is modelled through propositions. The only use of the temporal operators in CTL is to obtain a quantification over paths and states. A mapping will need to be performed so that the propositions used in the specifications can be evaluated at a states in different models, and the critical assumption is that this mapping will be very easy to specify.

3 From Sets of Worlds to Sets of Models

Several approaches in formal semantics take sentences to denote sets of worlds. For normative statements, we assume (following Kratzer [12]) that worlds are connected by an accessibility relation. Consider (1) in Section 1 which among other things *requires a test for Hepatitis B if no exceptions apply.* A denotation of this requirement is given in (3), and is the set of worlds w_0, such that for every deontically accessible world w, for every entity x such that x is a donation in that world $d'(x,w)$, if no exception holds of that donation $\neg e'(x,w)$, a test for Hepatitis B is carried out for that donation $\mathit{thepb}'(x,w)$. We will assume that negation has the highest precedence. Therefore $\neg a \to b \equiv (\neg a) \to b$, and brackets are used to resolve other ambiguities.

(3) $\lambda w_0.\forall w : (w \in \mathit{deo}(w_0) \to (\forall x : (d'(x,w) \to (\neg e'(x,w) \to \mathit{thepb}'(x,w)))))$

A difference between worlds in Kratzer's denotations and states in a model is that: in a state there is no notion of entities and relations between them. All that is available at a state s is the set of propositions which are true at that state $\Pi(s)$. To map (3) to a form that is useful for checking conformance, we need two assumptions.

First, we assume that regulation denotes the *set of models* that conform to it. Intuitively speaking, w_0 in (3) can be thought of as a model in its entirety, and $w \in \mathit{deo}(w_0)$ correspond to special states in the model. A universal quantification over accessible worlds can be replaced with the CTL AG operator. We then obtain the denotation in (4), read as : *on every path in M, if a state is deontically accessible, for each donation x at that state, if no exception holds, a test is carried out.* In a model, only special states (like when the bloodbank has finished processing all the donations it has received) need to conform to the regulation, and *deo* can be thought of as marking those states.

(4) $\lambda M.\ M \vDash AG(\mathit{deo} \to (\forall x : (d'(x) \to (\neg e'(x) \to \mathit{thepb}'(x)))))$

(4) is still not in CTL because of the universal quantification over entities x at a state. The universal quantifier can be eliminated by assuming *a*

serial processing model. This has the effect that at the deontically accessible states, exactly one donation is under consideration (e.g. the models in Figures 1(a) and 1(b)). In the sections of the CFR that we examined, a universal quantification over entities is absolutely essential when these entities correspond to inputs of an implemenation. This assumption lets us tie the inputs to states, and use the quantification over states to achieve the quantification over entities. Thus (4) can be reduced to (5).

(5) $\lambda M.\ M \vDash AG(deo \rightarrow (d' \rightarrow (\neg e' \rightarrow thepb')))$

A problem that is encountered in taking this approach is that there is no distinction between *obligations*, and *permissions* (both of which stem from the Hohfeldian legal conceptions of right, duty, privilege, and no right [9]). While this did not cause a problem for the obligation in (1), if one were to follow the same procedure for the permission in (2), we would get the denotation in (6).

(6) $\lambda M.\ M \vDash \neg(AG(deo \rightarrow (sp' \rightarrow thepb')))$

A model satisfies (6) only if *there is some path in which there is a state that is deontically accessible, and if a donation of source plasma is being processed it is not tested.* This is too strong a requirement, because an organization may choose not to do what it is permitted to do. The model in Figure 1(a) is a valid model, which would be declared invalid if (6) were required of it.

Another problem is that it is not clear how one would use (6) in interpreting the exemption e' in (5). A reasonable candidate is $e' \equiv deo \rightarrow (sp' \rightarrow \neg thepb')$. But this is not the exemption because it is true in *every deontically accessible state in which a donation of source plasma is not being processed.* Consider a state s at which $sp' = false$ $(sp' \notin \Pi(s))$. At s, $e' \equiv (deo \rightarrow (false \rightarrow \neg thepb')) \equiv (deo \rightarrow true) \equiv true$. The specification in (5), at s is: $AG(deo \rightarrow (\neg e \rightarrow \neg thepb')) \equiv AG(deo \rightarrow (\neg true \rightarrow \neg thepb')) \equiv AG(deo \rightarrow true) \equiv AG(true) \equiv true$. Therefore, a model that doesn't test any donation for Hepatitis B would conform to (5). We now turn to the task of addressing these problems by revising how the specifications are composed.

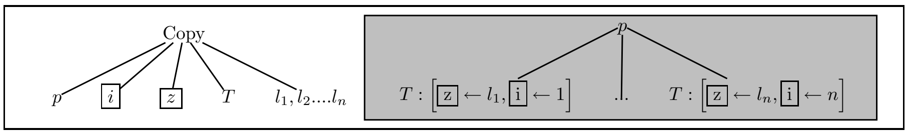
4 Extracting the specifications

To aid the requirements engineer in extracting the specifications, the idea is to present her with intermediate semantic representations of the sentence with which she interacts. The intermediate representations that we use fall into the category of *abstract syntax trees (ASTs).* ASTs are generally used as intermediate representations in compiling code in a high-level programming language to machine dependant code. The internal nodes in ASTs are *operators* (predicates/meta-predicates), the subtrees they dominate are *operands* (arguments), and leaf nodes correspond to variables or constants (the requirements engineer specifies the denotation of the leaves). An AST encodes the resolution of scope ambiguities, i.e., if p_1 dominates p_2 in the AST, then p_1 outscopes p_2.

Section 4.1 describes some phenomena in natural language that can be used in the construction of the ASTs, and how these ASTs can be interpreted. In Section 4.2, we describe how the ASTs and their interpretation for (1) and (2) (in Figures 3 and 4) address the problems described in Section 3. [3]

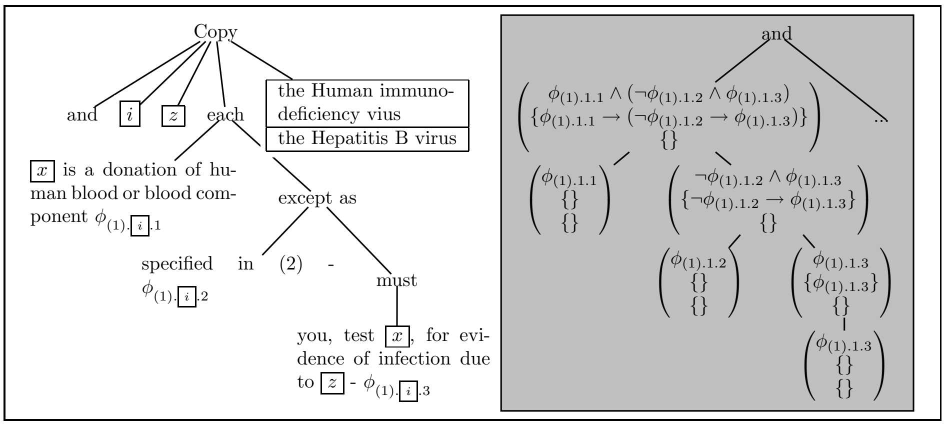
4.1 Abstract Syntax Trees (ASTs) and their interpretation

To capture the distinction between obligations and permissions, the denotation of each node N in an AST is given by the 3-tuple: $[[N]] = \begin{pmatrix} \phi_N \\ \mathcal{O}_N \\ \mathcal{P}_N \end{pmatrix}$, where $\mathcal{O}_N$ is a set of propositional logic formulas which correspond to *the obligations that have been satisisfied*, and $\mathcal{P}_N$ is a set of propositional logic formulas that correspond to *the permissions that have been taken*, and ϕ_N is a propositional logic formula which can be thought of as indicating whether N is true at a state. The set of obligations $\mathcal{O}$ obtained from the policy base is the union of the obligations obtained at the root of the AST for each sentence. The denotation of the policy base is then given by: $\lambda M.\ M \models AG\left(deo \rightarrow \bigwedge_{\phi \in \mathcal{O}} \phi\right)$. We now identify various linguistic constructions that can be used to obtain ASTs.



Copy
p
i
z
T
$l_1, l_2 .... l_n$
p
$T : [\boxed{z} \leftarrow l_1, \boxed{i} \leftarrow 1]$
...
$T : [\boxed{z} \leftarrow l_n, \boxed{i} \leftarrow n]$

Fig. 2. Semantics of the *Copy* meta-predicate



Copy
and
i
z
each
the Human immuno-deficiency vius
the Hepatitis B virus
$\boxed{x}$ is a donation of human blood or blood component $\phi_{(1).\boxed{i}.1}$
except as
specified in (2) - $\phi_{(1).\boxed{i}.2}$
must
you, test $\boxed{x}$, for evidence of infection due to $\boxed{z}$ - $\phi_{(1).\boxed{i}.3}$
and
$\phi_{(1).1.1} \wedge (\neg\phi_{(1).1.2} \wedge \phi_{(1).1.3})$
$\{\phi_{(1).1.1} \rightarrow (\neg\phi_{(1).1.2} \rightarrow \phi_{(1).1.3})\}$
$\{\}$
...
$\phi_{(1).1.1}$ $\{\}$ $\{\}$
$\neg\phi_{(1).1.2} \wedge \phi_{(1).1.3}$ $\{\neg\phi_{(1).1.2} \rightarrow \phi_{(1).1.3}\}$ $\{\}$
$\phi_{(1).1.2}$ $\{\}$ $\{\}$
$\phi_{(1).1.3}$ $\{\phi_{(1).1.3}\}$ $\{\}$
$\phi_{(1).1.3}$ $\{\}$ $\{\}$

Fig. 3. AST and its interpretation for (1)

Distributive readings and the *Copy* meta-predicate: (1) is ambiguous between *a collective reading* (where there is a single test for both the

[3] We assume that obligation and permission denoting categories, e.g. *must*, do not occur in contexts like antecedent clauses of subordinating conjunctions (like *if*), and restrictors of determiners. Handling these cases requires an extension to CTL which is beyond the scope of this paper.

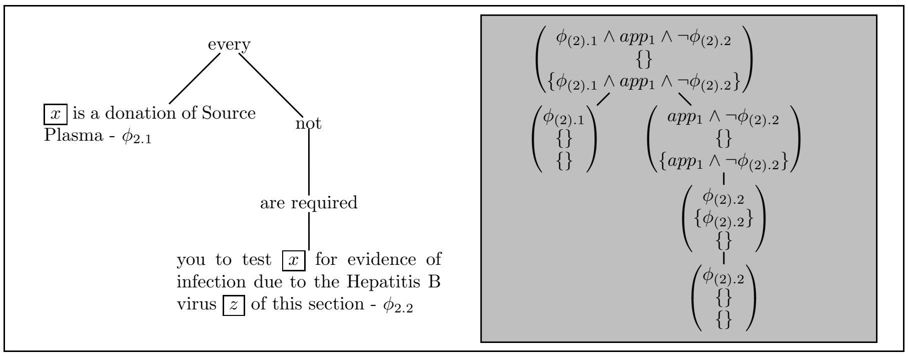

every
$\boxed{x}$ is a donation of Source Plasma - $\phi_{2.1}$
not
are required
you to test $\boxed{x}$ for evidence of infection due to the Hepatitis B virus $\boxed{z}$ of this section - $\phi_{2.2}$
$\phi_{(2).1} \wedge app_1 \wedge \neg\phi_{(2).2}$, $\{\}$, $\{\phi_{(2).1} \wedge app_1 \wedge \neg\phi_{(2).2}\}$
$\phi_{(2).1}$, $\{\}$, $\{\}$
$app_1 \wedge \neg\phi_{(2).2}$, $\{\}$, $\{app_1 \wedge \neg\phi_{(2).2}\}$
$\phi_{(2).2}$, $\{\phi_{(2).2}\}$, $\{\}$
$\phi_{(2).2}$, $\{\}$, $\{\}$

Fig. 4. AST and its interpretation for (2)

diseases), and *a distributive reading* (where there are separate tests for each disease). However, (2) gives an exemption to a test for one of the diseases, and this suggests that a distributive reading may be more appropriate in the specifications extracted, and that the distributivity has scope over the exception. Hence *Copy* dominates *except* in Figure 3.

The interpretation of the *Copy* meta-predicate is given in Figure 2. It is called a meta-predicate because it is a function from an AST to another AST, by simple variable substitution. For the AST for (1) shown in Figure 3, this results in an AST rooted with *and* with subtrees corresponding to each of the tests. The interpretation of *and* in this context is given by:

$$and \begin{pmatrix} \phi_A^1 \\ \mathcal{O}_A^1 \\ \mathcal{P}_A^1 \end{pmatrix} \ldots \begin{pmatrix} \phi_A^n \\ \mathcal{O}_A^n \\ \mathcal{P}_A^n \end{pmatrix} = \begin{pmatrix} \wedge_{i=1}^n \phi_A^i \\ \cup_{i=1}^n \mathcal{O}_A^i \\ \cup_{i=1}^n \mathcal{P}_A^i \end{pmatrix}$$

The RHS of the equation corresponds to the denotation of the node labeled *and* in the AST (shaded in gray in Figure 3).

Universally Quantified NPs correponding to inputs: As mentioned in Section 3, the universal quantification over inputs (donations) is achieved by associating states with unique inputs. The interpretation of the determiner *each* is designed with idea that the obligations will be evaluated at each state.

$$each \begin{pmatrix} \phi_A \\ \{\} \\ \{\} \end{pmatrix} \begin{pmatrix} \phi_B \\ \mathcal{O}_B \\ \mathcal{P}_B \end{pmatrix} = \begin{pmatrix} \phi_A \wedge \phi_B \\ \{\phi_A \rightarrow \phi^B_{\mathcal{O}.j} | \phi^B_{\mathcal{O}.j} \in \mathcal{O}_B\} \\ \{\phi_A \wedge \phi^B_{\mathcal{P}.j} | \phi^B_{\mathcal{P}.j} \in \mathcal{P}_B\} \end{pmatrix}$$

The interpretation of the determiner *no* is similar to that of *each/every*, except that a negation needs to be applied to the nuclear scope. We discuss the interpretation of negation in what follows.

Conditional and Exceptive constructions: There are several predicates that denote conditions and exceptions. For example, the subordinating conjunctions *if*, *unless*, and *except as*, coordinating conjunctions like *except that* or *but*. The interpretation of *if* is the same as that for *every*. The interpretation of predicates like *except as*, and *unless* are similar, the only difference being that $\neg\phi_A$ is used instead of ϕ_A in the RHS.

Modals and Negation: The semantics of modals and negation are given below:

$$must \begin{pmatrix} \phi_A \\ \{\} \\ \{\} \end{pmatrix} = \begin{pmatrix} \phi_A \\ \{\phi_A\} \\ \{\} \end{pmatrix} \quad may \begin{pmatrix} \phi_A \\ \{\} \\ \{\} \end{pmatrix} = \begin{pmatrix} app_i \wedge \phi_A \\ \{\} \\ \{app_i \wedge \phi_A\} \end{pmatrix}$$

$$not\begin{pmatrix}\phi_A\\ \mathcal{O}_A\\ \mathcal{P}_A\end{pmatrix}=\begin{pmatrix}\phi'_A\\ \{\neg\phi^A_{\mathcal{P}.j}|\phi^A_{\mathcal{P}.j}\in\mathcal{P}_A\}\\ \{app_j\wedge\neg\phi^A_{\mathcal{O}.j}|\phi^A_{\mathcal{O}.j}\in\mathcal{O}_A\}\end{pmatrix}\text{, where }\phi'_A=\begin{cases}app_j\wedge\neg\phi_A & \phi_A\equiv\phi^A_{\mathcal{O}.j}\in\mathcal{O}_A\\ \neg\phi_A & otherwise\end{cases}$$

$must(A)$ results in the interpretation that ϕ_A is an obligation. $may(A)$ results in the interpretation that $app_i \wedge \phi_A$ is a permission, where app_i is a variable introduced which the implementation must set to true when the permission is applied (we discuss its use in Section 4.2). And intuitively, the interpretation of negation captures the idea that $may(\neg A) \equiv not(must(A))$.

4.2 Discussion

There are two obligations obtained at the root of the AST for (1): $\phi_{(1).1.1} \rightarrow (\neg\phi_{(1).1.2} \rightarrow \phi_{(1).1.3}) \equiv d' \rightarrow (\neg e'_1 \rightarrow thiv')$ and $\phi_{(1).2.1} \rightarrow (\neg\phi_{(1).2.2} \rightarrow \phi_{(1).2.3}) \equiv d' \rightarrow (\neg e'_2 \rightarrow thepb')$, where d' is true iff the donation is one of blood or blood component, e'_1 and e'_2 are the exceptions to the required test for each disease, and $thiv'$ and $thepb'$ are true iff tests for HIV and Hepatitis B respectively have been performed. The computation of the second obligation is not shown in Figure 3, and is obtained from the second child of *and* (in the AST shaded in gray). Note that the individual propositions like d' need to be specified by the requirements engineer at the leaf nodes of the AST.

Figure 4 shows the AST and its interpretation for (2). The permission obtained at the root node is : $\phi_{(2).1} \wedge app_1 \wedge \neg\phi_{(2).2} \equiv sp' \wedge app_1 \wedge \neg thepb'$ where sp' is true iff a donation of source plasma is being processed, and $thepb'$ is true iff a test for the Hepatitis B virus has been carried out.

The use of the app_1 proposition is as follows. It is possible for the regulation to cancel the permission given in (2), but there may be several cases in which permission not to test a donation of source plasma for Hepatitis B is given. Suppose the case under consideration is one where the permission in (2) is cancelled, but the organization doesn't test a donation of source plasma for Hepatitis B because a different permission can be applied. Since the permission being applied sets $thepb'$ to false, and sp' is true, the only way for the implementation to indicate that the permission in (2) is not being applied is by setting app_1 to false. Setting $e'_1 \equiv false$, and $e'_2 \equiv sp' \wedge app_1 \wedge \neg thepb'$:

$$\phi_{\mathcal{O}.1} \equiv d' \rightarrow (\neg false \rightarrow thiv'),\text{ and }\ \phi_{\mathcal{O}.2} \equiv d' \rightarrow (\neg(sp' \wedge app_1 \wedge \neg thepb') \rightarrow thepb')$$

Considering just these obligations, the denotation of the regulatory document would be: $\lambda M.\ M \vDash AG(deo \rightarrow (\phi_{\mathcal{O}.1} \wedge \phi_{\mathcal{O}.2}))$. Therefore, a bloodbank could decide not to test a donation of source plasma for Hepatitis B, but they would always have to test a donation for HIV.

5 Conclusions and Future Work

We have described a framework to assist a requirements engineer in extracting CTL specifications from regulatory documents. An account of obligations and permissions turns out to be essential in composing the specifications. The composition procedure (defined in Section 4) was applied to a large part of the FDA CFR 610.40. While it does seem to scale well, providing tool support to extract and interact with the ASTs is vital. To this end, we plan to conduct a small scale annotation of ASTs which will let us determine the accuracy with which these representations can be computed. On the user interface side, we are working on ways of presenting the ASTs to the requirements engineer.

References

[1] Breuker, J. and N. den Haan, *Separating world and regulation knowledge: where is the logic?*, in: M. Sergot, editor, *Proceedings of the third international conference on AI and Law* (1991), pp. 41–51.

[2] Clarke, E. M. and E. A. Emerson, *Synthesis of synchronization skeletons for branching time temporal logic*, in: *Logic of Programs: Workshop*, 1981.

[3] Clarke, E. M., E. A. Emerson and A. P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems **8** (1986), pp. 244–263.

[4] Clarke, E. M. and J. M. Wing, *Formal methods: State of the art and future directions*, ACM Computing Surveys **28** (1996), pp. 626–643.

[5] Corbett, J. C., M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby and H. Zheng, *Bandera: Extracting finite-state models from java source code*, in: *Proceedings of the International Conference on Software Engineering (ICSE)*, 2000.

[6] Fantechi, A., S. Gnesi, G. Ristori, M. Carenini, M. Marino and M. Moreschini, *Assisting requirements formalization by means of natural language translation*, Formal Methods in System Design **4** (1994), pp. 243–263.

[7] Fuchs, N. and R. Schwitter, *Attempto controlled english (ace)*, in: *First International Workshop on Controlled Language Applications*, 1996.

[8] Glasse, E., T. V. Engers and A. Jacobs, *Power: An integrated method for legislation and regulations from their design to their use in e-government services and law enforcement*, in: M.-F. Moens, editor, *Digitale Wetgeving, Digital Legislation*, Die Keure Brugge, 2003 pp. 175–204, iSBN 90 5958 039 7.

[9] Hohfeld, W. N., *Fundamental legal conceptions as applied in judicial reasoning*, Yale Law Journal **23** (1913), pp. 16–59.

[10] Holt, A. and E. Klein, *A semantically-derived subset of English for hardware verification*, in: *37th Annual Meeting of the ACL*, 1999.

[11] Holzmann, G., *The Spin model checker*, IEEE Trans. on Software Engineering **23** (1997), pp. 279–295.

[12] Kratzer, A., *The notational category of modality*, in: H.-J. Eikmeyer and H. Rieser, editors, *Words, Worlds, and Contexts. New approaches to Word Semantics*, deGruyter, Berlin, 1981 .

[13] Queille, J. P. and J. Sifakis, *Specification and verification of concurrent systems in CAESAR*, in: *Proceeding of the Fifth ISP*, 1981.