Model Checking Legal Documents\textsuperscript{1}

Daniel GORÍN\textsuperscript{a} Sergio MERA\textsuperscript{a} Fernando SCHAPACHNIK\textsuperscript{a}
\textsuperscript{a}\{dgorin, smera, fschapachnik\}@dc.uba.ar
Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Abstract. This article presents the FORMALEX toolset, an approach to legislative
drafting that, based on the similarities between software specifications and some
types of regulations, uses off-the-shelf LTL model checkers to perform automatic
analysis on normative systems.

Keywords. Automated Legislative Drafting, Model Checking Regulations

1. Introduction

We propose a toolset to describe and analyze normative systems, using LTL and stan-
dard model checkers as core technologies. Its theoretical foundations are laid out in [2],
while a longer tool-based description is presented in [3]. FORMALEX provides a high
level language, FL, that not only allows to model the different normative propositions of
the normative system under analysis (NSUA) but also to express the background theory.
Besides the rules themselves, some extra information is usually needed to state the be-
aviours that are expected from the real world such as precedence of events (e.g., sunrise
before doom), uniqueness, etc. The background theory is where we encode such type of
information.

The normative propositions are represented by a set of LTL formulas, enriched with
deontic operators. FL treatment of obligation is classic: $O(\varphi)$ means that it must always
be the case that $\varphi$ holds and $F(\varphi)$ is just $O(\neg \varphi)$. There’s also the view of obligation
not as something that must always hold, but rather as something that must be done,
usually within some bound of time, sometimes called non-persistent obligation: “You
ought to return the borrowed books within three days”. We can also accommodate for
that using intervals inside obligations, as explained in the next section. CTD obligations
are supported as $O_p(\varphi)$, and $F_p(\varphi)$ is interpreted as $O_p(\neg \varphi)$. It is worth noticing that
although an ought-to-be approach is used, our encoding also skips out most of the deontic
logic paradoxes (see [2] for details).

A detail of notice is how we treat permissions: although we understand that permis-
sions as exceptions are of common use, we take a simpler perspective where a permis-
sion is a check that the rest of the rules must pass. That is, $F(kill)$ will not pass the test
$P(kill \text{ in self-defense})$. When analyzed, that would be outlined as a potential problem by
our tool. If corrected, then $F(kill \text{ unless self-defense})$ would indeed pass the check. For

\textsuperscript{1}Partially supported by PICT 2007 532, UBACyT 2008 X634.
space reasons we only postulate here that for a broad class of regulations this simple use of permission is sufficient, and refer to [2] for a deeper discussion on the topic.

Another difficult issue in deontic logic is the concept of coherence of a normative system: whenever the rules are “contradictory” in any sense. It is known ([4]) that the problem cannot be simply reduced to logical consistency. We take a pragmatic approach where a NSUA is not coherent if it has any of an explicit list of problems detailed in [2].

Other checks that can spot potential problems are performed, such as detecting the case of a rule stating $O(a \rightarrow b)$ and a background theory that allows $b$ to happen in any moment. Is it the expected behavior that $a$ is only a sufficient condition or it was also intended that it was a necessary condition?

2. Language Outline

We are going to show the most representative components of FL with a toy example instead of formal definitions. Let’s suppose we want to formalize a normative system that regulates the life cycle of a university student. The system describes under which conditions a student can take exams. It assumes a scenario where there is an academic year with two semesters per year. It also assumes that a student can take an exam only during a semester and that he can fail or pass the exam. These statements are not explicit in the document, and therefore the most natural thing is to specify this behavior in the background theory. First, we define two actions $YearBegins$ and $YearEnds$ that represent when the academic year begins and ends. Observe that these actions are always paired, that is, for each occurrence of $YearBegins$ there must be in the future an occurrence of $YearEnds$. Also while the academic year is passing by, there are no occurrences of any of those actions. In such a setting, we say that both actions form an interval, described as follows:

```
actions YearBegins, YearEnds
interval AcademicYear defined by actions YearBegins-YearEnds
```

Now we should say that there are two semesters per academic year. A semester also is defined as an interval of two actions, but they can only occur inside an academic year.

```
actions SemesterBegins, SemesterEnds
interval Semester defined by actions SemesterBegins-SemesterEnds
only occurs in scope AcademicYear occurrences 2
```

Finally, a student can take an exam only during a semester, and the result of that exam is pass or fail:

```
action TakeExam outputValues {Pass, Fail} only occurs in scope Semester
```

It is quite easy to see that these statements can be easily translated into the usual automata specification languages, but we omit the details for space reasons.

Let’s look now at the document itself. To model each of the document terms, we are going to use an extension of LTL designed for these purposes, including the usual LTL constructs plus the deontic operators previously mentioned.

Some remarks about the semantics of the actions in the context of a formula: when the action $a$ occurs, then the propositional symbol $a$ holds in that precise point in time.
and not further. On the other hand, a.\texttt{OutputValue} starts to hold in the point where a happens with output value \texttt{OutputValue}, and continues to hold until the action is out of scope or when a occurs again.

Let’s encode some of the document terms. “The students are obliged to take at least one exam per academic year” is $O(\Diamond_{\text{AcademicYear}} \text{TakeExam})$, where we use the obligation operator $O(\varphi)$, plus the operator $\Diamond_{\text{interval}} \varphi$ that says that there is a state in the future where $\varphi$ must hold but within the interval interval. An analogous case using the $F(\varphi)$ forbiddenness operator, “It is prohibited for a student to traverse a semester without taking exams” is encoded as $F(\Diamond_{\text{Semester}} \neg \text{TakeExam})$.

We translate $O_{\psi}(\varphi)$ as $\Box(\neg \varphi \rightarrow \psi)$. The rest of the legal operators are mapped to the latter: $O(\varphi) \equiv O_{\bot}(\varphi)$, $F(\varphi) \equiv F_{\bot}(\varphi)$ and $F_{\psi}(\varphi) \equiv O_{\psi}(\neg \varphi)$, where $\bot$ is the proposition that is always false. Finally, we assume that inside any interval AnInterval, delimited by its open-close actions, the proposition AnIntervalOpened holds (this can be easily accomplished when defining the set of traces). Therefore we translate $\Diamond_{\text{AnInterval}} \varphi$ as AnIntervalOpened $\rightarrow$ (AnIntervalOpened $U \varphi$). The translation of $\square_{\text{AnInterval}} \varphi$ is analogous.

Once specified, we perform various kind of coherence checks ([2]) by translating the formulas into plain LTL and performing specific checks on the set of traces of the NSUA, such as existence of at least one legal behaviour, subsumption of formulas, etc.

3. Case Study

We present a small case study where the use of formal analysis proved useful. It formalizes the regulations for the check-in process of an airline company and is inspired in a similar one from [1]. The legal document is as follows:

1. The ground crew is obliged to open the check-in desk two hours before the flight leaves.
2. After the check-in desk is opened the check-in crew is obliged to initiate the check-in process with any customer present by checking that the passport details match what is written on the ticket and that the luggage is within the weight limits. Then they are obliged to issue the boarding pass.
3. If the luggage weighs more than the limit, the crew is obliged to collect payment for the extra weight and issue the boarding pass.
4. The ground crew is prohibited from issuing any boarding cards without inspecting that the details are correct beforehand.
5. The ground crew is prohibited from issuing any boarding cards before opening the check-in desk.
6. The ground crew is obliged to close the check-in desk 20 minutes before the flight is due to leave and not before.
7. Once the check-in desk is closed, the ground crew is prohibited from issuing any boarding pass or from reopening the check-in desk.
8. The passenger must have a travel insurance. If this is not the case, she can buy one before she receives the boarding pass.

Our proposed encoding is as follows:

\begin{itemize}
\item \textbf{temporal actions} TwoHoursToDepart, TwentyMinutesToDepart
\item \textbf{actions} DeskOpens, DeskCloses, PaxLeaves
\item \textbf{action} PaxArrives \textbf{outputValues} \{WithInsurance, WithoutInsurance\}
\item \textbf{interval} Desk defined by \textbf{actions} DeskOpens-DeskCloses
\end{itemize}
interval Pax defined by actions PaxArrives-PaxLeaves only occurs in scope Desk
action PaxProvidesInfo occurrences 1 in scope Pax
   outputValues {Ok, Wrong} only occurs in scope Pax
action WeighLuggage occurrences 1 in scope Pax
   outputValues {Normal, Overweighted} only occurs in Pax
actions DeliverBoardingPass, ChargeFee, BuyInsurance max occurrences 1 in scope Pax only occurs in Pax

1. O(TwoHoursToDepart → DeskOpens)
3. O((PaxProvidesInfo.Ok∧WeighLuggage.Overweighted) → □ρνChargeFee∧
   O(PaxProvidesInfo.Ok ∧ ChargeFee) → □ρνDeliverBoardingPass)
4. F(!PaxProvidesInfo.Ok ∧ DeliverBoardingPass)
5. This item is already enforced by the background theory.
6. O(TwentyMinutesToDepart → DeskCloses)
7. This item is already enforced by the background theory.
8. Oρ(PaxArrives → PaxArrives.WithInsurance)∧
   F(DeliverBoardingPass∧PaxArrives.WithoutInsurance∧
   (occurred(BuyInsurance))) where ρ = □ρνBuyInsurance

When analyzed, the system passes most of the automatic checks, but one raises a warning in the action ChargeFee. The problem is that we have specified sufficient conditions to charge a fee, saying nothing about necessary conditions. This is reflected in ChargeFee only appearing as the consequence of a positive implication. The action ChargeFee could be performed even if the passenger has her luggage within the weight limit. Interestingly enough, this is not a problem of the way we encoded the normative system, but of the normative document itself. There should be an additional rule saying something like this:

9. The crew should collect a payment for the extra weight only when the passenger’s luggage weighs more than the limit. O(ChargeFee → WeighLuggage.Overweighted)

With this fix, now our normative system passes all the automatic checks. This show how our tool could help to solve a very common problem in normative systems.

References


2A temporal action occurs exactly once. The precedence between temporal actions corresponds to the declaration order.