12-2023

Deontics and time in contracts: An executable semantics for the L4 DSL

Seng Joe WATT
Singapore Management University, sjwatt@smu.edu.sg

Oliver GOODENOUGH

Meng Weng (HUANG Mingrong) WONG
Singapore Management University, mwwong@smu.edu.sg

Legal Knowledge and Information Systems
G. Sileno et al. (Eds.)

doi:10.3233/FAIA230954

Deontics and Time in Contracts: An Executable Semantics for the L4 DSL

Seng Joe WATT [a,1], Oliver GOODENOUGH [b] and Meng Weng WONG [a]
[a] *Singapore Management University Centre for Computational Law*
[b] *CodeX The Stanford Center for Legal Informatics and Vermont Law and Graduate School*

Abstract. Existing approaches to modelling contracts often rely on deontic logic to reason about norms, and only treat time qualitatively. Using L4, a textual domain specific language (DSL) for the law, we offer a more operational interpretation of norms, based on states and transitions, that also accounts for the granular timing of events. In this paper, we present a higher-level rendering of the loan agreement from Flood & Goodenough in L4, and an accompanying operational semantics amenable to execution and static analysis. We also implement this semantics in Maude and show how this lets us visualize the execution of the loan agreement.

Keywords. formal specification, contract automation, norm operationalization

1. Introduction

Thus far, much of the work on analysing and reasoning about norms has been facilitated by encoding them into deontic logic and logic programs, see for instance [1]. On the other hand, Flood & Goodenough [2] propose that many financial contracts are inherently computational in nature, and can be formalized via deterministic finite automata (DFA). This is demonstrated by encoding a simple loan agreement as a DFA, with states representing various situations contemplated in the contract and transitions between them denoting events triggering a change in situation. While this approach is a useful and important demonstration of how one can obtain an executable model of a contract, the DFA formalism has limitations. Perhaps the most apparent is that the DFA is arguably far-removed from a human's intuitive understanding of a contract, which is often expressed textually, via normative language. Finding a practical approach to move from the DFA to another executable formalism that addresses these issues is an obvious next step.

In this paper, we introduce the L4 DSL for law and show how its textual syntax enables concepts like deontics and deadlines to be expressed more naturally, in the style of a controlled natural language. We propose an operational semantics for L4 that allows us to execute contracts, and lays the foundation for future work on formal analysis. To

[1]Corresponding Author: sjwatt@smu.edu.sg. This research/project is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

illustrate the usefulness of this approach, we implement this semantics in Maude and apply this to visualize the loan agreement in [2].

Limitations of the DFA approach. While one can easily encode consequences of real-world events as transitions in a DFA as in [2], it is common in many legal traditions to reason about and specify contracts in terms of normative requirements that need to be fulfilled by various actors [1]. We believe that normative language embodies the psychology that many humans bring to questions of contractual compliance, with such language acting as a small "widget" for what could be broken out into a chain of smaller steps around a logic of event and consequence. L4's syntax serves some of this purpose.

Contracts also frequently embody some notion of time and deadline. These domain-specific concepts are an integral part of contract drafting, as are the notions of actors and actions. Unfortunately, these are not primitives of DFAs, so they must be encoded manually, and in ways that may appear unnatural to a lay reader. For example, the passage of time is usually explicitly reflected as an achieved event.

2. The L4 approach

The L4 language [3,4] for encoding legal texts addresses these shortcomings, using a natural language-based structure and syntax which allows the detailed, if artificially structured, expression of the obligations and consequences that need to be set out in most legal specifications, at a higher level than the DFA formalism allows. L4, as with LegalRuleML [5], distinguishes between two kinds of rules: (i) constitutive, and (ii) regulative (or prescriptive) rules. The former encodes static decision logic, which are similar to Horn clauses in logic programs. The focus of this work is on the latter, i.e. regulative rules, which concern the norms specifying expected actions in a legal text. These are a central component in the computational structure of contracts like the loan agreement. They govern the conditions and situations in which an actor is obliged, permitted or prohibited from performing an action, as well as the consequences of fulfilling and violating the given norms.

Syntax of regulative rules. Below is a EBNF grammar of the syntax of regulative rules in L4. Here we use the following metavariables: `actor`, `action` and `ruleName` range over strings, while n ranges over all natural numbers.

$$\langle \text{rule} \rangle \to \langle \text{regulative} \rangle \mid \texttt{ruleName}\ \text{“}\texttt{MEANS}\text{”}\ \langle \text{ruleNames} \rangle$$
$$\langle \text{regulative} \rangle \to (\S \mid \text{“}\texttt{RULE}\text{”})\ \texttt{ruleName}\ \text{“}\texttt{PARTY}\text{”}\ \texttt{actor}\ \langle \text{deontic} \rangle\ \texttt{action}\ \langle \text{deadline} \rangle\ [\langle \text{henceClause} \rangle]\ [\langle \text{lestClause} \rangle]$$
$$\langle \text{deontic} \rangle \to \text{“}\texttt{MUST}\text{”} \mid \text{“}\texttt{MAY}\text{”} \mid \text{“}\texttt{SHANT}\text{”}$$
$$\langle \text{deadline} \rangle \to (\text{“}\texttt{ON}\text{”} \mid \text{“}\texttt{WITHIN}\text{”})\ n\ \text{“}\texttt{DAY}\text{”}$$
$$\langle \text{henceClause} \rangle \to \text{“}\texttt{HENCE}\text{”}\ \langle \text{ruleNames} \rangle$$
$$\langle \text{lestClause} \rangle \to \text{“}\texttt{LEST}\text{”}\ \langle \text{ruleNames} \rangle$$
$$\langle \text{ruleNames} \rangle \to \{\texttt{ruleName}\ \text{“}\texttt{AND}\text{”}\}\ \texttt{ruleName}$$

Briefly, the intended meaning of a regulative rule is that the `actor` `MUST/MAY/SHANT` perform the `action` according to the `deadline`. For deontics, `MUST`, `MAY` and `SHANT`, denote obligations, permissions and prohibitions respectively. Note that these deontic modalities have specific characteristics which are not fully in alignment with their use in other systems. `henceClause` and `lestClause` specify the consequences of fulfilling and violating the rule, in terms of which new rules that get triggered, i.e. become active.

Deadlines are relative to the time when the rule is triggered, with `WITHIN` *n* `DAY` (resp. `ON` *n* `DAY`) indicating that the action may occur at any time within the next *n*-days (resp. only on the *n*-th day).

The rule names in a `henceClause` (resp. `lestClause`) denote rules that are triggered when the obligation or prohibition is fulfilled (resp. violated). We identify the fulfilment of an obligation with whether the actor performed the prescribed action within the deadline, and violation is identified with the deadline passing before the actor performs the action. These are defined symmetrically for prohibitions. Permissions can only be fulfilled and not violated, and rules in the `henceClause` (resp. `lestClause`) are triggered when the permission is exercised (resp. not exercised within the deadline). Finally, the `MEANS` construct is used to combine rules, so that triggering the rule to the left of the `MEANS` effectively triggers all those to its right.

The loan agreement in L4. Currently, the input format of L4 is comma-separated values (CSV) rendered from a spreadsheet, with Google Sheets forming the primary interface for interacting with L4. The L4 encoding of the contract can be found here [2].

```
§ Contract Commencement
 PARTY Borrower
   MAY request funds
WITHIN 1 DAY
 HENCE Remit principal

§ Remit principal
 PARTY Lender
  MUST remit principal
WITHIN 1 DAY
 HENCE Repayment

       Repayment
 MEANS Repay in two halves
   AND Avoid default
```

```
§ Repay in two halves
 PARTY Borrower
  MUST repay first half
    ON 5 DAY
 HENCE Repay second half
  LEST Notify borrower

§ Accelerated repayments
 PARTY Borrower
  MUST repay outstanding amount
WITHIN 1 day
```

Figure 1. Excerpt from L4 loan agreement

Fig. 1 contains an excerpt, which we use here to give an intuition for our operational, state-transition based semantics. As a starting point, our encoding of the loan agreement uses a version of the agreement that has been changed in some provisions from [2] as we currently lack certain features like fixed dates, which are used as deadlines there. To cope with this, we specify deadlines in terms of days relative to the time the rule triggers. And for simplicity, we chose to use a shorter deadline of 5 days for the repayments.

For simplicity, we assume that the first rule in an L4 specification is the starting rule, so that contract execution begins with only this rule being active. No other rules in the L4 specification are active at this point, though they may get triggered as part of the `henceClause` or `lestClause` of the starting rule, and of other rules later during execution. In this case, the starting rule is labelled `Contract Commencement`, which says that the actor, `Borrower`, may perform the `request principal` action any time from the time the rule comes into effect up til 1 day later. The `henceClause` indicates that if the permission to make a loan is exercised, another rule called `Remit Principal`

[2] https://docs.google.com/spreadsheets/d/1_k17mzk2hTyPY2egbEVqXzr8uYAdi6BVqeMYdBxKB_I/edit#gid=864296175

is triggered. The lack of a `lestClause` indicates that no new rule is triggered when the deadline passes without the permission being exercised. This obliges the lender to send the principal amount to the borrower once the borrower has requested it; but if no such request is made, the contract terminates successfully. Currently, we assume that permissions are used up once the action occurs, so that no repeated actions are allowed. This assumption mirrors that of [6], in which it is noted that one can also allow for permissions that may be exercised more than once.

The `Repayment` rule, which represents the main body of the contract, is defined via `MEANS`, and is triggered when the `Remit principal` rule is fulfilled, i.e. when the lender sends the principal to the borrower. Triggering this `Repayment` rule then has the effect of triggering the `Repay in two halves` and `Avoid default` rules in parallel. The former obliges the borrower to make both repayments sequentially, and the latter prohibits him from defaulting on representations and warranties.

As in [7], we distinguish between the violation of an obligation/prohibition and the violation of the contract as a whole. Obligations and prohibitions like `Remit principal`, which do not have a `lestClause`, are assumed to be non-compensable. This means that when this obligation is violated, we deem the contract to have been breached by the lender. Execution of the contract proceeds no further, so that execution terminates in a breach state.

In contrast, some rules, like the `Repay in two halves` obligation, are compensable, with compensation mechanisms appearing in the `lestClause`. In this case, the violation of the rule results in the rules in the `lestClause` being triggered, and the execution of the contract continues, as opposed to terminating in a breach state.

In the `Repay in two halves` rule, the `ON 5 DAY` deadline disallows early payments, as in the original agreement. The `Notify borrower` rule grants the lender permission to notify the borrower and encodes the "notice and cure" compensation mechanism. As noted in [2], this indicates the start of the compensation pathway, which gives the borrower a second chance, by obliging the borrower to cure their default. Should the borrower still not make amends, the execution of the contract has reached the point of no return and no further second chances are given. This is embodied by the `Accelerated repayments` obligation being triggered.

Unfortunately, our encoding of this rule does not faithfully capture the intuitive reading of its natural language counterpart. That is, when the execution of the contract reaches this stage, all other existing rules are irrelevant, and so one would like to terminate them all when this comes into effect. While we would have liked to add support for this, we encountered various complications and so left this for future work.

3. Transition system semantics and implementation

To define our semantics, we use the mathematical framework of Structural Operational Semantics (SOS) [8], to describe a transition systems in terms of states, called *configurations*, and *transition rules* that describe how one state can transition to another.

Definition [Configuration, Active rule instance] A configuration C is a pair of either: (i) $(\texttt{Active}, R)$ where `Active` indicates that the contract is still active (i.e. not breached), and R, called the set of *active rule instances*, contains pairs $(\texttt{ruleName}, t)$ where t is a timer tracking the remaining to fulfill (resp. exercise) the obligation/prohibition (resp.

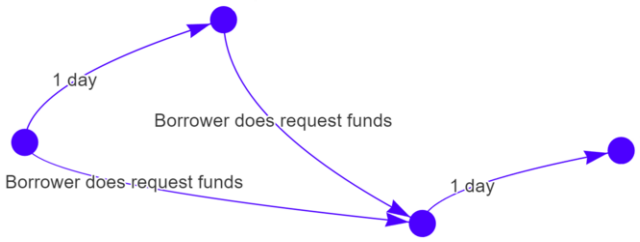

Figure 2. Start of contract

permission), or (ii) $(\texttt{Breached}, A)$, a breach state, where A is the set of actors who breached the contract.

The above definition for configurations, and that of our transition rules, are inspired by timed transition systems [9]. Configurations track rules that are active at each point during execution, along with their corresponding deadline. When a rule r is triggered, we place a new active rule instance (r, t) into R, with t being the initial deadline of r.

For transitions, we distinguish between: (i) *action transitions*, and (ii) *tick transitions*, which denote that an actor performed an action, and the passage of one day respectively. Each of these has an associated *transition function*, $\delta_{\text{action}}(\texttt{actor}, \texttt{action}, C)$ and $\delta_{\text{tick}}(C)$, defined recursively on R to compute the effect of the transition on the current configuration C. Due to space constraints, we describe these informally here and refer the interested reader to [10] for details. Informally, δ_{action} removes active rule instances where the corresponding rule of the instance has the same actor and action as the action which occurred, triggering new rules in `henceClause` or `lestClause` appropriately. δ_{tick} decrements timers t of active rule instances and triggers new rules similarly to $\delta_{\texttt{action}}$. In their definitions, we ensure that if a non-compensable rule, i.e. a `MUST` or `MAY` rule without a `lestClause` is violated, δ_{action} and δ_{tick} yield a breach state $(\texttt{Breached}, \{\texttt{actor}\})$ where `actor` is the actor who violated the rule. With this, we have the following transition rules in our SOS:

$$\frac{\text{applicable}(C, \texttt{actor}, \texttt{action})}{C \xrightarrow{(\texttt{actor},\texttt{action})} \delta_{\text{action}}(\texttt{actor}, \texttt{action}, C)}\ \text{action} \qquad \frac{}{C \xrightarrow{1\ \text{day}} \delta_{\text{tick}}(C)}\ \text{tick}$$

$\text{applicable}(C, \texttt{actor}, \texttt{action})$ is used to restrict action transitions to pairs of $(\texttt{actor}, \texttt{action})$ which appear in rules that currently have active rule instances. Our semantics does not currently allow for one to prioritize or override rules, nor does it account for deontic inconsistencies. This means that for instance, configurations are allowed to contain active rule instances of say, a `MUST` and `SHANT` rule that refer to the same actor and action. Such a configuration remains active, and execution continues as normal. Analysing a contract for such inconsistencies is planned for future work.

Executing and visualizing with Maude. We implemented our semantics in Maude [11], a language grounded in equational and rewriting logic. With Maude, we can naturally define: (i) our transition functions, δ_{action} and δ_{tick}, and (ii) our transition system with rewriting rules mirroring our SOS. Such a transition system can then be executed and model checked using tools such as [12]. While our current work does not account for deontic inconsistencies, we aim to explore using such tools to detect them in the future.

We have implemented tools to transform L4 CSV specifications into a plain text format for consumption by Maude, which we then used to execute specifications and visualize the state space. Fig. 2 shows the start of the loan agreement in the generated state

space, with the leftmost state being the starting one. Action transitions are labelled with the corresponding actor and action e.g. `Borrower does request funds` , while tick transitions are labelled `1 day`. Hovering over each state reveals text with more details. For the initial state, we have the following, with the `1 DAY` indicating the remaining time to fulfill the obligation:

```
Active ( RULE Contract Commencement PARTY Borrower MAY request
funds WITHIN 1 DAY HENCE Remit Principal )
```

4. Conclusion

In this paper, we report on our explorations on operationalizing regulative rules from a model contract in L4. We began with the DFA approach of [2] and moved to the more natural language like formalism that we currently have in L4. We demonstrated how L4 allows us to naturally express key domain concepts like deontics and deadlines, which are found in these rules. We opted for a timed transition system semantics and showed how this can be implemented in Maude, in order to visualize contract execution. Future work includes addressing the limitations mentioned throughout the paper, as well as integrating Maude's model checking facilities to detect deontic inconsistencies.

References

[1] Robaldo L, Batsakis S, Calegari R, Calimeri F, Fujita M, Governatori G, et al. Compliance checking on first-order knowledge with conflicting and compensatory norms: a comparison among currently available technologies. Artificial Intelligence and Law. 2023 Jun.

[2] Flood MD, Goodenough OR. Contract as automaton: representing a simple financial agreement in computational form. Artificial Intelligence and Law. 2021 Oct.

[3] CCLAW. L4 Documentation;. Accessed: 2023-10-26. https://l4-documentation.readthedocs.io/en/latest/index.html.

[4] Mahajan A, Strecker M, Wong MW. Overview of the CCLAW L4 project. In: POPL 2022 - Programming Languages and the Law. Philadelphia, United States; 2022. .

[5] Palmirani M, Governatori G, Rotolo A, Tabet S, Boley H, Paschke A. LegalRuleML: XML-Based Rules and Norms. In: Olken F, Palmirani M, Sottara D, editors. Rule-Based Modeling and Computing on the Semantic Web. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. p. 298-312.

[6] Chircop S, Pace G, Schneider G. In: An Automata-Based Formalism for Normative Documents with Real-Time; 2022. .

[7] Governatori G, Rotolo A. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. Australasian Journal of Logic. 2006 09;4:193-215.

[8] Plotkin GD. A Structural Approach to Operational Semantics. University of Aarhus; 1981. DAIMI FN-19.

[9] Henzinger TA, Manna Z, Pnueli A. Timed transition systems. In: de Bakker JW, Huizing C, de Roever WP, Rozenberg G, editors. Real-Time: Theory in Practice. Berlin, Heidelberg: Springer Berlin Heidelberg; 1992. p. 226-51.

[10] Deontics and time in contracts: An executable semantics for the L4 DSL;. Accessed: 2023-10-27. https://www.researchgate.net/publication/375025000_Deontics_and_time_in_contracts_An_executable_semantics_for_the_L4_DSL.

[11] Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, et al., editors. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. vol. 4350 of Lecture Notes in Computer Science. Springer; 2007.

[12] Rubio R, Martí-Oliet N, Pita I, Verdejo A. Strategies, model checking and branching-time properties in Maude. Journal of Logical and Algebraic Methods in Programming. 2021;123:100700.