Formalizing a simple loan agreement in L4 and Maude

Joe Watt

March 22, 2023

This is a work in progress.

Contents

1 Intro and shortcomings of DFA formalization

1.1 Intro

TODO
Write a proper introduction to the original work on formalizing the simple loan agreement as a DFA.

In [2], Flood and Goodenough propose that many financial contracts are inherently computational in nature. They argue that the computational structure of many such contracts can be formalized via deterministic finite automata (DFAs), with states representing various situations. Transitions between these states then correspond to events triggering a change in these situations. This is demonstrated using a simple loan agreement [2], which they formalize directly as a DFA.

1.2 Shortcomings

While this approach is a useful and important proof of concept, as the authors themselves recognize, there are some limitations with such a formalism, perhaps the most apparent being that the manual encoding of a contract as a DFA is a laborious process, and one that in itself does not produce a machine executable result One source of complexity arises from the fact that the DFA is a low level computational formalism that is arguably far removed from our intuitive understanding of a contract. While we can easily encode real-world events as transitions in a DFA, as done in [2], some legal traditions reason about and specify contracts in terms of normative requirements that need to be fulfilled by various actors [3]. The authors of this paper, representing different traditions, do not entirely agree on this point, but it remains true that a DFA is a relatively mechanistic chain of event and consequence, and is not adept at normative representation.

Contracts also frequently embody some notion of time and deadline. These are examples of domain-specific concepts which are an integral part of contract drafting. Unfortunately, DFAs have no primitive notion of these, so that these must all be encoded manually, with the passage of time reflected as an achieved event, in a manner that is perhaps unnatural to many.

Another source of complexity is the state explosion problem. Complex contracts have large DFA representations and are thus impractical for humans to specify manually at scale. Observe that a simplified visual representation of the automaton [2, Fig. 1] corresponding to this simple contract already contains more than 20 states and 40 transitions.

One of the main causes of this is the concurrent interleaving of real world events. By this, we mean that many real world events can occur in any order (including at the same time) and perhaps for the sake of simplicity, this has not been accounted for in the DFA formalization. As an example of such a scenario, consider the following. On May 31, 2015, the day before payment 1 is due, the borrower defaults on his representations and warranties. On the next day, when payment 1 becomes due, the borrower diligently repays that payment. One day later, on June 2, 2015, the lender notifies the borrower of his earlier default, which does not get cured after another 2 days. Now all outstanding payments become accelerated, and the borrower pays off the remaining amount of $525 in time, causing the contract to terminate. This sequence of events, when viewed as a word over the event alphabet, is unfortunately not accepted by the automaton.

Closer inspection of the DFA suggests that there is an implicit assumption being made that once an event of default occurs, the borrower will be notified by the lender soon after, with no other events like payments occurring in between.

More formally, it is assumed that the default and notification events occur within the same atomic step. Splitting this up into separate events would increase the number of states in the DFA, especially if we account for concurrency. In fairness, Flood and Goodenough understood some of these limits, and specified in [2][Section 6] of their model agreement that: "In the event of multiple events of default, the first to occur shall take precedence for the purposes of specifying outcomes under this agreement". This provision goes some way toward resolving competing defaults by the expedient of mandating that later events will simply be disregarded.

2 The L4 approach

2.1 L4 and the CCLAW project

At CCLAW, we are developing one such domain specific language (DSL) for specifying legal contracts, called L4, which addresses these issues. Along with it, we are also developing an accompanying toolset which can be used to visualize and analyze contracts written in L4. Here we first give an introduction to our project and the L4 DSL that we are developing, before discussing our formalization of the loan agreement in the next section.

As described in , our focus is on rule-based reasoning, as opposed to case-based reasoning. Following LegalRuleML , we distinguish between two kinds of rules, namely constitutive and regulative (or prescriptive) rules. The former encodes static decision logic, similar to Prolog-style *if-then* rules, and is treated as such in our implementation.

Our focus here is on the latter, namely regulative rules, which concern the norms in a legal text. These tell us under which conditions an actor is allowed, permitted or prohibited from performing an action, as well as the consequences of fulfilling and violating them. While deontic logic approaches have been popular, our approach to regulative rules in L4 is closer in spirit to [1, 4], in that our treatment of the rules yields a state transition system, similar in spirit to the DFA found in [2].

Cite WAICOM 2022 and PROLALA 2022 papers

Cite LegalRuleML

Insert example, like the one in PROLALA paper

Cite appropriately

Cite the Maude paper as well

2.1.1 Syntax of regulative rules

In L4, users can encode regulative rules textually, using a syntax that looks like a highly restricted form of controlled natural language. These have the following form:

```
RULE rule name
PARTY actor
deontic
[ deadline ]
action
[ HENCE hence_0 AND ... AND hence_n ]
[ LEST lest_0 AND ... AND lest_n ]
```

Note that the deadline, as well as the `HENCE` and `CLAUSES` are optional.

Each clause in the above rule corresponds to the following domain concepts that we have identified:

- `actor`
- `action`

  Actors and actions are currently plain strings with no special meaning attached to them.

- `deontic`

  There are 3 kinds of deontics:

  - `MUST`: Achievement obligation
  - `MAY`: Permission
  - `SHANT`: Prohibition

- `deadline`

  This can take the form:

  - `WITHIN` n `DAY`
  - `ON` n `DAY`

  These are relative to the time when the rule comes into effect and `WITHIN` n `DAY` indicates that the action may occur at any time within the next n days, while `ON` n `DAY` means that it may only occur on the n-th day itself.

- `hence_i` (resp `lest_i`)

  `hence_i` (resp `lest_i`) are rules that are triggered when the obligation or prohibition is fulfilled (resp violated).

  In the case of permissions, the `hence_i` are triggered when the permission is exercised, and the `lest_i` are triggered when it is not exercised by the deadline.

2.2 The loan agreement in L4

(TODO: Add link)

The following rule is found at the beginning in our encoding of the loan agreement. In L4, we assume the first rule at the top of the specification to be the starting rule that is triggered once the contract begins.

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

The intuitive meaning of such a rule is 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. Note that for simplicity, we assume that permissions are one-off and is consumed once the action occurs, so that no repeated actions are allowed. This assumption mirrors that of [1], in which the author notes that one can also allow for permissions that may be exercised more than once. In the future, we hope to lift this restriction and add support for permissions that can be exercised more than once before the deadline.

The `HENCE` clause here indicates that if the permission is exercised, then another rule called `Remit Principal` comes into effect. The lack of a `LEST` clause indicates that no new rule is triggered when the deadline passes without the permission being exercised.

The `Remit principal` rule is then expressed as follows:

```
RULE Remit principal
PARTY Lender
MUST remit principal
     in the amount of 1000
     to       Borrower
WITHIN        1       DAY
HENCE Repayment
```

This obliges the lender to send the principal amount to the borrower once

3 L4 through Maude

In search for such a suitable formalism for encoding contracts, we have surveyed various approaches, using the simplified contract in [2, Fig 1.] as an example. One of these which we explored is the Maude language and its associated tools.

In this section, we begin by providing some background on Maude before we discuss our formalization of the simple loan agreement. Along the way, we explain how we addressed the aforementioned shortcomings. In particular, we show how we can generate an arguably more accurate DFA that accounts for the concurrent interleaving of real world events. Thereafter, we demonstrate how we can use Maude to help us visualize and reason about the contract.

3.1 Background on Maude

Techniques originating from the field of formal methods have been devised to automatically analyze the behavior of computer systems. These often rely on first modelling these systems as *labelled transition systems* (LTS), which can be seen as generalizations of nondeterministic finite automata). As with finite automata, LTSes can be seen as directed graphs, with nodes representing states that the system can be in, and labelled edges denoting transitions that change the state of a system. Where they differ from finite automata is that they are allowed to have an infinite (possibly uncountable) number of states and transitions between them. They are also not required to come with a notion of initial and final states.

While DFAs and LTSes in general are formalisms that are well suited for computers to analyze and reason about, they are cumbersome for humans to use to encode systems. This is especially so for systems like the simple loan agreement of [2] which involve concurrency, in that there are many possible ways in which events can be interleaved. Modelling concurrent systems like this directly as a transition system is often impractical as the interleaving of events gives rise to numerous states and transitions. Moreover, as previously discussed, they leave implicit many of the domain specific concepts that people directly reason about.

Maude is a language that allows us to model and analyze such concurrent systems. As with others like it, it comes equipped with a more sophisticated formalism that allow us to more conveniently specify these transition systems.

3.2 Translating from L4 to Maude

Use the loan agreement as an example here.

3.3 Visualizing and simulating the contract

Insert DFA of state space here.

4 Related work

5 Limitations and future work

The current Maude models are not very performant and have huge state spaces. One of the main reasons is that we model time naively, with a `tick` transition denoting the passage of 1 unit of time (which can be taken to be 1 day, 1 month etc). We did this because Maude does not come with any built-in support for these notions, so we wanted an easy way to model these. Unfortunately, this considerably bloats up the state space. Again it must be emphasized that this is a proof of concept and these were designed to be as high-level and close to the syntax of L4 as possible. In the future, we intend to optimize this, among other things.

Currently, we don't yet support global variables and more sophisticated control structures like loops. Global variables would be useful to capture the notion of the outstanding amount in the loan agreement contract, which would then vary with the payments made by the borrower. Maude as a formalism supports these, but we currently do not support them in L4. For future work, we would like to provide syntax for these in L4 and translate them to Maude.

References

[1] Chircop, S., Pace, G., and Schneider, G. *An Automata-Based Formalism for Normative Documents with Real-Time*. 12 2022.

[2] Flood, M. D., and Goodenough, O. R. Contract as automaton: representing a simple financial agreement in computational form. *Artificial Intelligence and Law* (Oct 2021).

[3] Hashmi, M., Governatori, G., and Wynn, M. Normative requirements for regulatory compliance: An abstract formal framework. *Information Systems Frontiers 18* (05 2015).

[4] Sasdelli, D. Normative diagrams. In *Proceedings of the International Workshop on AI Compliance Mechanism (WAICOM 2022)* (12 2022), pp. 39–49.