Formalizing a simple loan agreement in mcrl2

Joe Watt

September 15, 2022

This is a work in progress.

Contents

1 Intro and shortcomings of DFA formalization

1.1 Intro

TODO:
Write a better intro and provide more background on [Flood and Goodenough, 2021].

In [Flood and Goodenough, 2021], the authors claim 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 [Flood and Goodenough, 2021, Table 1], which they formalize directly as a DFA.

1.2 Shortcomings

While this approach is a nice proof of concept, there are various shortcomings with such a formalism, perhaps the most apparent being that the manual encoding of a contract as a DFA is a laborious process. A simplified, arguably inaccurate, visual representation of the automaton [Flood and Goodenough, 2021, Fig. 1] corresponding to this simple contract already contains more than 20 states and 40 transitions. Here, we discuss 2 reasons contributing to the complexity involved with a more accurate formalization of the contract as a DFA. We claim that even for such a simple contract, a more accurate formalization as a DFA is too impractical to be carried out by hand.

The first source of complexity is a consequence of DFAs not having an explicit notion of global variables, as well as their inability to perform arithmetic computations. To see this, observe how there are verbose duplication of "Payments ... accelerating" states. Note here that when we say "duplication", we do not mean that there are states which play exactly the same role from the point of view of bisimilarity or language acceptance as in the sense of the Myhill-Nerode technique of DFA minimization. What we mean instead is that these states play similar roles, and viewing the DFA as a graph, the subgraphs rooted at these states have similar structure.

Viewing these subgraphs as sub-automata, we see that they both accept similar sequences of events from the point when the borrower defaults and is obliged to make an accelerated repayment of the outstanding amount. Notice that the key difference between them is the "Payment made \$$n$" transition, corresponding to the event that the borrower pays off the outstanding amount of \$$n$. While we would like to collapse both of these subgraphs into a single one, this is not possible as the value of n varies at runtime, with the current state of the contract, ie with the number of payments the borrower has paid off previously. In other words, n can be thought of as a global variable whose actual value is updated at runtime, whenever payment events occur. However, DFAs do not have a notion of global state, nor can they perform sophisticated computations like arithmetic ones, so that these values must be manually computed by hand and then encoded in the DFA. The result of this manual encoding is the aforementioned duplication of states and transitions. Now, if the contract were to contain more than just 2 repayment stages, more manual computations would be required, resulting in even more duplicated states and transitions that would then need to be added to the DFA.

The second source of complexity arises from the concurrent interleaving of real world events. Perhaps for the sake of simplicity, this has not been accounted for in the DFA formalization. As an example of such a scenario that is not handled by the DFA, 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*. One could argue that the real world is inherently concurrent and asynchronous in nature, so that a more realistic encoding of this contract would account for such interleaving of events. Of source, doing so would significantly increase the complexity of the DFA, and consequently, the labor involved with its manual construction.

Therefore, we argue that this evinces that accurately encoding contracts as a DFA directly is too laborious and difficult, even for a contract as simple as the one presented in [Flood and Goodenough, 2021, Fig 1.]. In our view, a more practical formalism should sit at a higher-level than a DFA, providing mechanisms for tackling these 2 issues. Firstly, it should provide global variables, or at least, some way of encoding global state. There should also be operations that allow us to retrieve and update the global state. Secondly, it should be able to conveniently accommodate the concurrency inherent in the real world. Better still if the formalism comes with tools that allow us to compile down to something like automata for visualization and automated reasoning.

2 Our approach

In search for such a suitable formalism for encoding contracts, we have surveyed various approaches, using the simplified contract in [Flood and Goodenough, 2021, Fig 1.] as an example. One of these which we explored is the `mcrl2` toolset [Bunte et al., 2019], which provides a high-level modeling language based on a process algebra [Groote and Mousavi, 2014].

In this section, we begin by providing some background on `mcrl2`. Thereafter, we discuss our formalization of the simple loan agreement and how it relates to the DFA as in [Flood and Goodenough, 2021]. In particular, we show how we can generate a DFA that accounts for the concurrent interleaving of real world events. Finally, we demonstrate how we can use the `mcrl2` toolset to help us reason about the contract.

TODO:
May also want to cover other related approaches, like Symboleo and how they use nuxmv for verification. Need to investigate how they model that further first.
Should mention limitations of our approach, the most obvious being that we don't handle real-time.

2.1 Background on mcrl2

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 (NFA). 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 [1] 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 [Flood and Goodenough, 2021] 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. As a result, many tools like the one we use, ie `mcrl2`, come equipped with more sophisticated formalisms that allow us to more conveniently specify these transition systems. Given a specification expressed in the provided formalism, these tools automatically generate the corresponding LTS, which they can then analyze.

The formalism provided by the `mcrl2` toolset is a textual specification language based on a process algebra, extended with real-time and data [Groote and Mousavi, 2014].

TODO:
More details about mcrl2 and some example code illustrating basic concepts.

[1] Possibly uncountable

2.2 The formalization of the contract itself

2.3 Background on the modal mu calculus in mcrl2

2.4 Automated reasoning with the contract

2.5 Limitations and future work

References

[Bunte et al., 2019] Bunte, O., Groote, J. F., Keiren, J. J. A., Laveaux, M., Neele, T., de Vink, E. P., Wesselink, W., Wijs, A., and Willemse, T. A. C. (2019). The mcrl2 toolset for analysing concurrent systems. In Vojnar, T. and Zhang, L., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–39, Cham. Springer International Publishing.

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

[Groote and Mousavi, 2014] Groote, J. F. and Mousavi, M. R. (2014). *Modeling and Analysis of Communicating Systems*. The MIT Press.