Overview of the CCLAW L4 project

Avishkar Mahajan, Martin Strecker, and Meng Weng Wong

Singapore Management University

Abstract

L4 is a domain-specific specification language that facilitates semantically rigorous formalization of legal expressions found in legislation and contracts. In our talk we will demo the working pieces (as of late 2021 / early 2022): a low-level core and a high-level language, a transpiler to reasoning back-ends including static analysis / formal verification and an expert system web app, and a real-world case study.

Keywords: Knowledge representation and reasoning, Argumentation and law, Computational Law, Defeasible reasoning, Programming language theory, Specification languages

1 Context

Computerized support for legal reasoning has a long history going back to the 1970s; in the decades since, a wide variety of approaches have appeared at venues such as ICAIL and JURIX. This paper shares the main tenets of our approach at the Centre for Computational Law[1] at Singapore Management University. We describe some principles and requirements in this section, and then give an overview of current work (Section 2).

The first principle focuses our work on rule-based reasoning in law (*vs* case-based reasoning). This approach lies in the formalist tradition, closer to pure civil law than pure common law. We explore the deductive and abductive modes of reasoning, leaving inductive reasoning (supported by similarity metrics) to the very active field of machine learning. The second principle is that law is meant to describe a set of admissible behaviours, and not a single behaviour, so our framework is declarative, in contrast to *smart contract* languages which are in fact programming languages. We seek to combine formal rigour with user-friendliness and accessibility to lay legal workers. This has led us to conceive our own domain-specific language (DSL) called L4, instead of adopting an existing general-purpose language. Finally, the language aims at supporting a variety of tasks relevant in a law context, ranging from legal drafting and reasoning *about* rules (such as consistency checking of a rule set) through compliance checking of business processes to reasoning *with* rules in order to derive consequences of an individual usage scenario.

L4 is inspired by prior research at Gothenburg, at Copenhagen, at Data61, at INRIA, by the LegalRuleML working group, and by many others.

2 Current Work

2.1 Language

We will now highlight some aspects and usage patterns of the L4 platform. We borrow from RuleML the notions of *constitutive* rules which provide ontology, and *prescriptive* rules which regulate behaviour. We first describe static prescriptive rules and their possible uses, then dynamic prescriptive rules for time-dependent, multi-agent processes.

[1] https://cclaw.smu.edu.sg/

In their simplest form, in the core language, **rules** are *if-then* statements, where both the antecedent and consequent are expressions. We encode an excerpt of the Professional Conduct Rules from Singapore's Legal Profession Act:[2]

```
rule <r1a>
for lpr: LegalPractitioner, app: Appointment
if (exists bsn : Business. AssociatedWithAppB app bsn && IncompatibleDignity bsn)
then MustNotAcceptApp lpr app
```

The core **expression** language here is a simply-typed lambda calculus. As seen in the example, there is no *a priori* restriction to using complex (quantified, higher-order, ...) expressions in rules, as long as they are well typed, but further processing steps may be restricted to fragments of the rule language or require previous (automated) transformations. With this, the above style of rule can be used without further ado in a Prolog-like fashion, to represent constitutive rules and simple prescriptive rules based on them.

In legal formalization, questions of deontics and of defeasible reasoning are inevitable. L4 supports neither of them in the core language, *i.e.* in the logic, but provides mechanisms for expressing them by other means. In dynamic, state-based system specifications, individual states can be flagged as breach states, allowing for a fine-grained reasoning about traces that avoid, reach or repair these breaches.

Neither is there native support for *default reasoning*. As an extra-logical feature, a rule can be marked with modifiers like *subject to* or *despite*, which prioritizes rules and provides a *defeasible* reasoning pattern that, in absence of knowledge to the contrary, a given rule is applicable. In a similar spirit, negation by failure can be simulated by rule inversion. It is worth noting that these mechanisms do not rely on an extension of the logic, but on compilation of rules.

The core language is augmented by a higher-level language, whose syntax is inspired by data-centric languages, which transpiles to the core and to other target languages. The following fragment, based on Singapore's Personal Data Protection Act[3] and related Regulations[4] expresses a prescriptive rule – actually three sub-rules, connected by the `HENCE` and `LEST` keywords:

```
 EVERY  Organisation  ("You")
UNLESS  the Organisation is a Public Agency
  UPON  becoming aware a data breach may have occurred
WITHIN  30 days
  MUST  assess
        if   it is a Notifiable Data Breach
  LEST      PARTY  Personal Data Protection Commission  ("PDPC")
          WITHIN  30 days
             MAY  demand
                  an explanation   for your inaction
           HENCE  You MUST  respond
                            to   PDPC
                    BEFORE  28 days
```

2.2 Back-End Services

We provide **reasoning services** by translating rules to other languages or solvers. According to the principle of usability for the target audience, we aim at a push-button technology with the following, mostly complementary, services:

- **SMT solvers:** Their purpose is to permit reasoning about a rule set and thus to explore conditions of consistency, completeness or other user-defined criteria. In the above example of professional conduct rules, one might be interested in knowing whether

[2] https://sso.agc.gov.sg/SL/LPA1966-S706-2015
[3] https://sso.agc.gov.sg/Act/PDPA2012?ProvIds=P1VIA-#pr26C-
[4] https://sso.agc.gov.sg/SL/PDPA2012-S64-2021?WholeDoc=1

`MustNotAcceptApp` and `MayAcceptApp` are complete and mutually exclusive. It has to be noted that the law text explicitly defines both predicates, so one cannot simply be assumed to be the negation of the other. For verification, users can write down a proof obligation (a formula). The L4 tool set translates this formula and the rules (eventually transformed, as mentioned above) to expressions in the the SMTLIB format, sends them to a SMT solver and retrieves the result. SMT solvers are in principle also capable of reasoning about specific scenarios, but the following methods are more appropriate.

- **Answer Set Programming:** Answer Set Programming (ASP) is a declarative programming language used widely in Knowledge Representation and Reasoning to model rules, facts, integrity constraints etc, within various domains. Using ASP, we want to derive legal conclusions from facts and rules and also justify those conclusions all while taking into account unique features of legal reasoning such as reasoning with priorities, exceptions etc. We also want to explore ways to automatically generate questions to solicit information from the user based on the input legal rules. The two ASP systems we are experimenting with are sCASP[5] and Clingo[6]. sCASP is of particular interest for us due to its ability to generate justification trees for conclusions. In Clingo, we have experimented with the use of a backward chaining and abductive search procedure for the purpose of automatically generating Boolean questions to ask the user. We are currently able to transpile a small subset of L4 rules into the required ASP programs to achieve some of the desired functionalities.
- **"Expert system":** By this, we understand a rule based production system where rules are applied to data stored in a fact base to derive new data, and this until a fixed point is reached. We currently experiment with the systems Drools[7], Clara[8] and O'Doyle[9]. As compared to primarily logical methods, the strength of these engines lies in their capability to process larger quantities of data.

Our current work concentrates on providing readable justifications for traces of rule applications or models produced by solvers. This includes a **visualization** component for justifications.

An essential element of the L4 environment is **natural language processing** provided through interaction with the Grammatical Framework[10]. Whereas the L4 language is a DSL in the programming language tradition, it contains natural language elements (see the second example rule) that are mapped to expressions of the core language (as in the first example rule). Beyond that, natural language processing and generation are essential in interactive legal advisors.

2.3 Tool support

The reader should be aware that most of what has been described is still work in progress, with an open-source development in Haskell available on CCLAW's Github.[11]

Acknowledgements. The work reported above is the result of the contributions of the whole CCLAW team (see the Github page). This research is supported by the National Research Foundation (NRF), Singapore, under its Industry Alignment Fund – Pre-Positioning Programme, as the Research Programme in Computational Law. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

[5] https://cliplab.org/papers/iclp2018-scasp.pdf
[6] https://potassco.org/clingo/
[7] https://www.drools.org
[8] http://www.clara-rules.org/
[9] https://github.com/oakes/odoyle-rules
[10] http://www.grammaticalframework.org/
[11] https://github.com/smucclaw