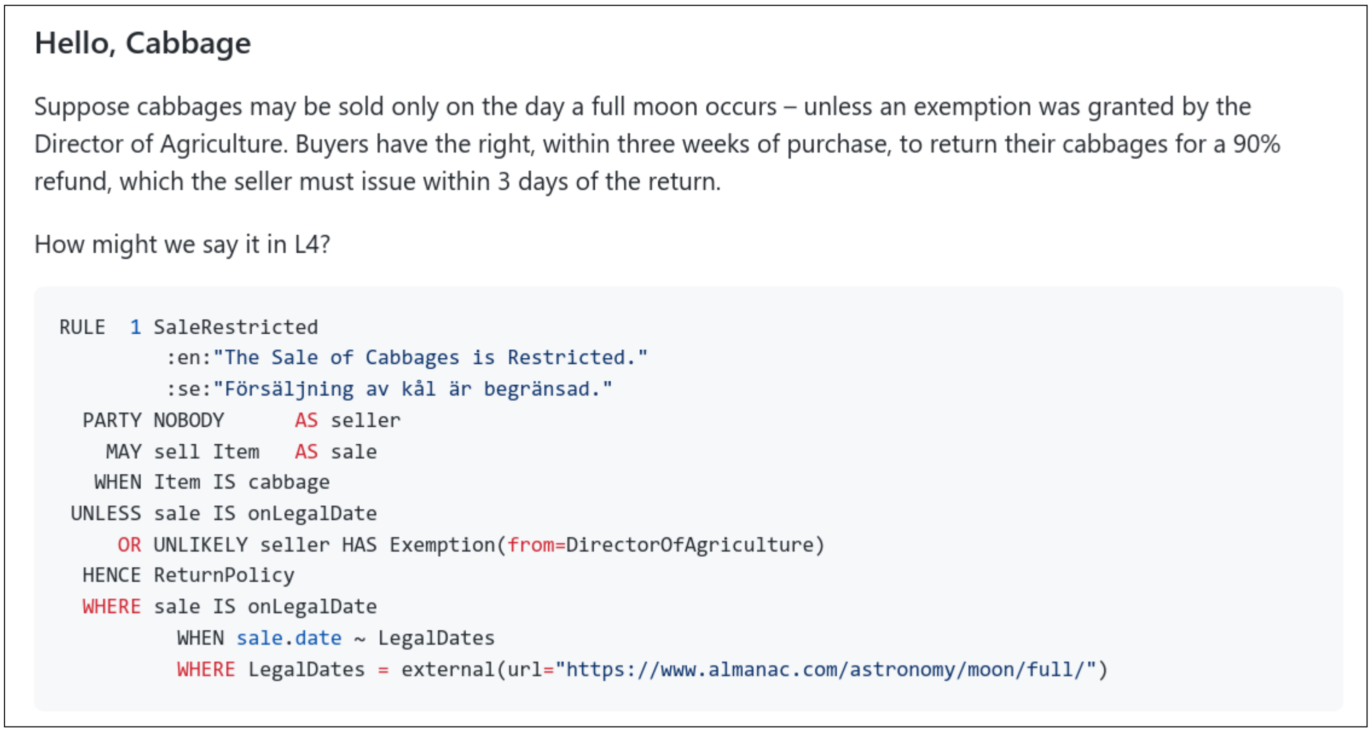
CS2309 Research Proposal: L4

Yee Jian Tan
yeejian.tan@u.nus.edu
National University of Singapore
Singapore



Hello, Cabbage

Suppose cabbages may be sold only on the day a full moon occurs – unless an exemption was granted by the Director of Agriculture. Buyers have the right, within three weeks of purchase, to return their cabbages for a 90% refund, which the seller must issue within 3 days of the return.

How might we say it in L4?

```
RULE  1 SaleRestricted
         :en:"The Sale of Cabbages is Restricted."
         :se:"Försäljning av kål är begränsad."
  PARTY NOBODY      AS seller
    MAY sell Item   AS sale
   WHEN Item IS cabbage
 UNLESS sale IS onLegalDate
     OR UNLIKELY seller HAS Exemption(from=DirectorOfAgriculture)
  HENCE ReturnPolicy
  WHERE sale IS onLegalDate
          WHEN sale.date ~ LegalDates
          WHERE LegalDates = external(url="https://www.almanac.com/astronomy/moon/full/")
```

Figure 1: The L4 Language. Taken from https://github.com/smucclaw/dsl/ on 13 November 2021.

ABSTRACT

The L4 language is a domain-specific language (DSL) designed to encode legal texts. On top of that, it is designed to analyze and reason with legal texts via SMT (and ASP) solvers. However, since it is a language still in development, there has not been mathematical verification of its completeness and consistency as a law DSL. This research proposal aims to sketch a plan using logic, automata and type theory as directions to investigate and formally verify this newly-formulated language.

CCS CONCEPTS

• **Software and its engineering** → *Constraint and logic languages*; **Domain specific languages**; • **Theory of computation** → **Logic and verification**; *Formal languages and automata theory*; • **Applied computing** → **Law**.

KEYWORDS

computational law, type theory, temporal logic

1 INTRODUCTION

This research proposal is inspired by, and builds on the work of SMU Center of Computational Law (CCLAW) [13, 14].

Legal documents are essentially a series of logical rules. They state conditions to be fulfilled with their consequences and alternates, no different from a typical *if-then-else* clause in programming. Formulating law into computer programs have been an active area of research, and multiple approaches have been taken. In particular, the SMU CCLAW has decided to write a Domain Specific Language (DSL), called L4, which encodes law text and provides a way to "compile" into natural language. More importantly, this DSL should be designed for analysis and verification in mind. Are there any conditions which is impossible to fulfill? Are there contradicting conditions which make certain rules ambiguous or redundant? To answer these questions, our DSL should allow analysis from they way it is designed.

L4, the language by SMU CCLAW, can allow different ways of analysis, including Satisfiable Modulo Theory (SMT) and Answer Set Programming (ASP). As a DSL to encode legal texts, we have to ensure that the language is complete and consistent. Complete as in it can encode any legal truth as stated in the documents, consistent as in the elimination rules of the semantics of the L4 language conforms to a subset of the natural deduction and hence is not self-contradictory.

2 PROBLEM AND PROPOSAL

We ask the question: **is the L4 language complete and consistent?** Although the authors should have paid sufficient attention to this regard, it is nonetheless paramount that the completeness and consistency of the system be **proven formally and mathematically**, especially for a domain as important as legal formalization. As of the time of writing, there are no such proofs to be found in the papers as provided in their Github repository [13]. I propose to answer this problem in 2 methods: first by exploring related **automata and logical systems** that are similar to that of law, secondly proving the inherent completeness and consistency of the system by the angle of **type theory**.

2.1 Automata and logical systems

In the approach to look at existing logic and automaton, we explore what are the logic systems in use for computational law, most likely systems such as Linear Temporal Logic, Buchi (Temporal) Automaton, as well as deontic logic and defeasible reasoning, which are prevalent in law ([8], [6]). Understanding how these theories fit with the L4 language would enable an inspection in the capabilities and differences between the logic of L4 and any selected system, and any significant differences or contradictions would signify a need to change the specifications of the L4 language.

2.2 Type Theory

The L4 language uses the concept of classes (or types), so it is natural to analyze this hierachy of type-induced propositions in the language. Type Theory refers to the study of typed lambda calculus systems such as Martin-Lof Type Theory and Homotopy Type Theory, and their correspondence to mathematical logic. The use of types in languages inherently gets its "proof strength" from the corresponding logical systems [2]. Existing logical systems in proof assistants such as Coq are not designed for the *temporal* way of reasoning in law, where an action done in different time can lead to different consequences. There are existing systems which aim to bring temporal logic to Coq, however implementations differ and it is not yet standardized [3, 10]. Another system of importance is Temporal Type Theory, which is known to already embed Linear and Metric Temporal Logic [11], fits the use case of L4 well.

3 RELATED WORKS

The Catala language [9] is a "programming language for law", one of the most prominent DSL for law. The compilation of Catala is verified by the F^* theorem prover. Catala has a multi-step compilation process, first translating from the concrete syntax to a desugared *scope language*, which is in turn translated into a lambda calculus-like *default calculus*, before compiling to a programming language such as OCaml that has runtime exceptions [9].

The approach to code legal reasoning with type theory so the facts can be proven mathematically is not new. LogiKEy workbench [1] uses the Isabelle interative proof assistant to code and prove legal rules, differing from the approach taken by L4 to use SMT and Answer Set Programming (ASP) solvers to automatically prove theorems [13]. Even so, the integration between a DSL for law and a proof assistant would be worth investigating, and is important to our research since proving facts about legal texts are a main part of what lawyers do.

As for logic, deontic logic and defeasible reasoning are two important starting points to explore the completeness of the L4 system [13]. Deontic logic is concerned about notions such as “the least one can do”, “obligatory”, “permissible” and other similar concepts, which often occur in law [8]. Deontic logic is emphasized in systems such as Symboleo [12] and the NAI Suite [7]. On the other hand, defeasible reasoning is about reasonings that are “rationally compelling but not deductively valid”, ie., when the premises are true but the derived conclusion is false. A combination of both deontic logic and defeasible reasoning forms the base of the Turnip system [4, 5], developed by Governatori and colleagues.

4 METHODOLOGY

To explore completeness of the existing L4 system, we will investigate by encoding a small piece of law or legal contract in L4 to explore the completeness requirements. Then by exploring existing systems as well and considering both deontic logic and defeasible reasoning, another survey on temporal logic and automata to match these features would be the end goal of the completeness investigation.

The type theory approach has two non-overlapping directions of investigation: one to explore extending L4’s type system to temporal type theory [11]. Then it will strengthen the L4 language, such that it is possible to prove theorems during the encoding of law into L4. However, this requires care as L4 currently allows users to define their own types, and self-define types might lead to significant overhead in the SMT and ASP solvers due to these unintended types-as-propostitions. Another approach would be to use a theorem prover such as Coq to prove the consistency of the axiomatic system and elimination rules of L4. This is a requirement before L4 can be more than merely an encoding of law like a controlled natural language, like an intermediate representation for SMT and ASP solvers to work on.

5 CONCLUSION

The task to prove the completeness and consistency of the L4 language can be challenging due to its open-endedness, but many related works already reached varying levels of success in formalizing law. While their logical systems can be used as a reference point, L4 is unique in its design to use SMT and ASP solvers from the start. In this case, the type theory approach is an important complementary way to show these important properties of this language, especially via the promising Temporal Type Theory.

REFERENCES

[1] Christoph Benzmüller, Ali Farjami, David Fuenmayor, Paul Meder, Xavier Parent, Alexander Steen, Leendert van der Torre, and Valeria Zahoransky. 2020. LogiKEy workbench: Deontic logics, logic combinations and expressive ethical and legal reasoning (Isabelle/HOL dataset). *Data in Brief* 33 (2020), 106409. https://doi.org/10.1016/j.dib.2020.106409

[2] Thierry Coquand. 2018. Type Theory. In *The Stanford Encyclopedia of Philosophy* (Fall 2018 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[3] Gmalecha. 2015. GMALECHA/coq-temporal: An implementation of temporal logic in COQ using charge! https://github.com/gmalecha/coq-temporal

[4] Guido Governatori. 2011. On the Relationship between Carneades and Defeasible Logic. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law* (Pittsburgh, Pennsylvania) *(ICAIL ’11)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/2018358.2018362

[5] Guido Governatori and Francesco Olivieri. 2021. Unravel legal references in defeasible deontic logic. *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law* (2021).

[6] Robert Koons. 2021. Defeasible Reasoning. In *The Stanford Encyclopedia of Philosophy* (Fall 2021 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[7] Tomer Libal and Alexander Steen. 2019. The NAI Suite - Drafting and Reasoning over Legal Texts. In *Legal Knowledge and Information Systems - JURIX 2019: The Thirty-second Annual Conference, Madrid, Spain, December 11-13, 2019 (Frontiers in Artificial Intelligence and Applications, Vol. 322)*, Michal Araszkiewicz and Víctor Rodríguez-Doncel (Eds.). IOS Press, 243–246. https://doi.org/10.3233/FAIA190333

[8] Paul McNamara and Frederik Van De Putte. 2021. Deontic Logic. In *The Stanford Encyclopedia of Philosophy* (Spring 2021 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[9] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. 2021. Catala: A Programming Language for the Law. In *ICFP 2021*. https://www.microsoft.com/en-us/research/publication/catala-a-programming-language-for-the-law/

[10] NotBad4U. 2019. NOTBAD4U/coqlang_temporal_logic: Formalization of temporal logic in Coq. https://github.com/NotBad4U/coqlang_temporal_logic

[11] Patrick Schultz and David I. Spivak. 2017. Temporal Type Theory: A topos-theoretic approach to systems and behavior. arXiv:1710.10258 [math.CT]

[12] Sepehr Sharifi, Alireza Parvizimosaed, Daniel Amyot, Luigi Logrippo, and John Mylopoulos. 2020. Symboleo: Towards

a Specification Language for Legal Contracts. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 364–369. https://doi.org/10.1109/RE48521.2020.00049

[13] Smucclaw. [n. d.]. Smucclaw/DSL: Domain specific languages for Computational Law. https://github.com/smucclaw/dsl/

[14] Singapore Management University. 2021. Home: Centre for computational law: Singapore Management University. https://cclaw.smu.edu.sg/