Towards a Sound and Complete Library for Modal Logics Reasoning in Coq

Ariel Agne da Silveira
Paulo Torrens
Karina Roggia
Rodrigo Ribeiro

ABSTRACT

bla bla

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Modal logics, soundness, completeness, Coq proof assistant

ACM Reference Format:
Ariel Agne da Silveira, Paulo Torrens, Karina Roggia, and Rodrigo Ribeiro. 2018. Towards a Sound and Complete Library for Modal Logics Reasoning in Coq. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Our society depends heavily on computing technology: our cars, airplanes, medical devices and banking applications all rely on software artifacts. In this scenario, *formal methods* play a crucial role to ensure that a program follows its specification, thus avoiding failures which can cause castrophic outcomes. Interactive proof assistants, like Coq [3] and Isabelle/HOL [12], can be used to verify software components and have been applied with success in non-trivial projects like a certified compiler for C programming language [9] and the kernel of seL4 operating system [8].

Usually, complex software systems are formed by various intercommunicating components. A key aspect of the design of such programs is how to ensure that a failure of a single component does not compromise the whole system. The main difficulty is such formalization tasks is how to guarantee correct error detection and that its corresponding fix will ensure that the software sill continue to work under abnormal conditions. One way to express such requirements is using *modal logics*, which can specify communication patterns and knowledge in software agents [7].

While modal logics allow the modelling of such requirements, proving properties about them may be a hard task without proper tool support. There are some dedicated provers for modal logics [4, 10], but we believe that using mature tools, like a proof assistant, are a better option, for verifying facts using modal logics. We justify this choice by: 1) the recent successful histories on verification of software and mathematics [5, 6, 8, 9]; 2) There is a growing community of developers using such tools thanks to textbooks on the subject [1, 2, 11] which allows newbies to join the game of formal verification and 3) The custom automation facilities that current proof assistants support allows that some trivial facts can be checked using decision procedures.

In this context, the present work aims to contribute by constructing a library that formalize a family of modal logics and its related properties in Coq. By using such a library, a developer could use all Coq's resources to prove modal logics facts.

More specifically, we contribute:

- We define a *deep embedding* of modal logic syntax in Coq and define its Kripke semantics.
- We model the Hilbert deductive systems K, D, B, T, K4, K5, S4 and S5 in Coq and prove ...
- We prove a lot of things...

The rest of this work is organized as follows: Section 2 provides brief introduction on modal logics and the Coq proof assistant. Related work is discussed on Section 3 and draws Section 4 draws some conclusions and provide some pointers for future works.

2 BACKGROUND

An overview of Coq proof assistant. Coq is a proof assistant based on the calculus of inductive constructions (CIC) [13], a higher order typed λ-calculus extended with inductive definitions. Theorem proving in Coq follows the ideas of the so-called "BHK-correspondence"[1], where types represent logical formulas, λ-terms represent proofs [1] and the task of checking if a piece of text is a proof of a given formula corresponds to checking if the term that represents the proof has the type corresponding to the given formula.

However, writing a proof term whose type is that of a logical formula can be a hard task, even for very simple propositions. In order to make the writing of complex proofs easier, Coq provides *tactics*, which are commands that can be used to construct proof terms in a more user friendly way.

As a tiny example, consider the task of proving the following simple formula of propositional logic:

$$(A \to B) \to (B \to C) \to A \to C$$

In Coq, such theorem can be expressed as:

[1]Abbreviation of Brouwer, Heyting, Kolmogorov, de Bruijn and Martin-Löf Correspondence. This is also known as the Curry-Howard "isomorphism".

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

```
Section EXAMPLE.
   Variables A B C : Prop.
   Theorem example : (A -> B) -> (B -> C) -> A -> C.
   Proof.
       intros H H' HA. apply H'. apply H. assumption.
   Qed.
End EXAMPLE.
```

In the previous source code piece, we have defined a Coq section named `EXAMPLE`[2] which declares variables `A`, `B` and `C` as being propositions (i.e. with type **`Prop`**). Tactic **`intros`** introduces variables `H`, `H'` and `HA` into the (typing) context, respectively with types `A -> B`, `B -> C` and `A` and leaves goal `C` to be proved. Tactic **`apply`**, used with a term `t`, generates goal `P` when there exists `t: P -> Q` in the typing context and the current goal is `Q`. Thus, **`apply`** `H'` changes the goal from `C` to `B` and **`apply`** `H` changes the goal to `A`. Tactic **`assumption`** traverses the typing context to find a hypothesis that matches with the goal.

We define next a proof of the previous propositional logical formula that, in contrast to the previous proof, that was built using tactics (**`intros`**, **`apply`** and **`assumption`**), is coded directly as a function:

```
Definition example :(A -> B) -> (B -> C) -> A -> C :=
 fun (H : A -> B)
     (H' : B -> C)
     (HA : A) => H' (H HA).
```

However, even for very simple theorems, coding a definition directly as a Coq term can be a hard task. Because of this, the use of tactics has become the standard way of proving theorems in Coq. Furthermore, the Coq proof assistant provides not only a great number of tactics but also a domain specific language for scripted proof automation, called $\mathcal{L}$tac. Details about $\mathcal{L}$tac and Coq can be found in [1–3].

An overview of modal logic. Modal logics are designed for reasoning about truth across various — abstract — worlds. In such logics, a proposition may be true in some world, but false in another one. The versions of modal logics considered in this work extends traditional propositional logics with two operators on propositions: $\Box$ (box) and $\Diamond$ (diamond). The syntax of modal logic formulas are defined by the following context free grammar (where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $\mathcal{P}$ is the enumerable set of propositional variables and $A \in \mathcal{P}$ is arbitrary):

$$\varphi \quad ::= \quad \bot \mid \top \mid A \mid \neg\varphi \mid \varphi \circ \varphi \mid \Box\varphi \mid \Diamond\varphi$$

Intuitively, the semantics of the operator $\Box$ is similar to an universal quantifier, since it express *necessity*: $\Box\,\varphi$ is true at the current world iff φ is true at all worlds. Similarly, the semantics of operator $\Diamond$ is that a formula $\Diamond\varphi$ is true at the current world iff φ is *possible*, i.e. true in at least some world.

Formally, we can only interpret a formula with respect to a *model*, which is defined as follows. A *frame* $\mathcal{F} = \langle W, R\rangle$ consists of a pair formed by a non-empty set of worlds, W, and a binary accessibility relation between worlds, $R \subseteq W \times W$. A pair $w_1 R w_2$ denotes that world w_2 is accessible from w_1. A model $\mathcal{M} = \langle \mathcal{F}, \mathcal{V}\rangle$ of a frame $\mathcal{F}$ and a total labeling function $\mathcal{V} : \mathcal{P} \times W \longrightarrow \{F, T\}$, which assigns a truth value to a propositional variable at a given world, i.e. when $w \in \mathcal{V}(A)$ for some $A \in \mathcal{P}$, we say that A holds in w. The satisfiability of a formula in a given world is defined by inductive relation $\mathcal{M}\,;\,w \models \varphi$ as follows (notation $\mathcal{M}\,;\,w \not\models \varphi$ denotes that $\mathcal{M}\,;\,w \models \varphi$ does not hold):

(1) $\mathcal{M}\,;\,w \not\models \bot$ and $\mathcal{M}\,;\,w \models \top$;
(2) $\mathcal{M}\,;\,w \models A$ iff $\mathcal{V}(A, w) = T$;
(3) $\mathcal{M}\,;\,w \models \neg\varphi$ iff $\mathcal{M}\,;\,w \not\models \varphi$;
(4) $\mathcal{M}\,;\,w \models \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}\,;\,w \models \varphi_1$ and $\mathcal{M}\,;\,w \models \varphi_2$;
(5) $\mathcal{M}\,;\,w \models \varphi_1 \vee \varphi_2$ iff $\mathcal{M}\,;\,w \models \varphi_1$ or $\mathcal{M}\,;\,w \models \varphi_2$;
(6) $\mathcal{M}\,;\,w \models \varphi_1 \rightarrow \varphi_2$ iff $\mathcal{M}\,;\,w \not\models \varphi_1$ or $\mathcal{M}\,;\,w \models \varphi_2$;
(7) $\mathcal{M}\,;\,w \models \varphi_1 \leftrightarrow \varphi_2$ iff $\mathcal{M}\,;\,w \models \varphi_1 \rightarrow \varphi_2$ and $\mathcal{M}\,;\,w \models \varphi_2 \rightarrow \varphi_1$;
(8) $\mathcal{M}\,;\,w \models \Box\varphi$ iff $\forall w \in W. wRy \rightarrow \mathcal{M}\,;\,y \models \varphi$;
(9) $\mathcal{M}\,;\,w \models \Diamond\varphi$ iff $\exists w \in W. wRy \wedge \mathcal{M}\,;\,y \models \varphi$;

We say that a formula φ is satisfiable in a model $\mathcal{M}$, denoted by $\mathcal{M} \models \varphi$, if $\forall w \in W. M\,;w \models \varphi$ holds. Similarly, φ is valid in a frame $\mathcal{F}$, written $\mathcal{F} \models \varphi$, if it holds on all models $\mathcal{M} = \langle \mathcal{F}, \mathcal{V}\rangle$, i.e. for any $\mathcal{M}$, we have that $\mathcal{M} \models \varphi$. Finally, we say that a formula is valid, written $\models \varphi$, if it is valid in any frame $\mathcal{F}$.

3 RELATED WORK

4 CONCLUSION

REFERENCES

[1] Yves Bertot and Pierre Castran. 2010. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions* (1st ed.). Springer Publishing Company, Incorporated.
[2] Adam Chlipala. 2013. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant.* The MIT Press.
[3] The Coq development team. 2019. *The Coq proof assistant reference manual.* LogiCal Project. http://coq.inria.fr Version 8.9.0.
[4] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. 2017. Theorem provers for every normal modal logic. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning.* EasyChair, 14–30.
[5] Georges Gonthier. 2008. *The Four Colour Theorem: Engineering of a Formal Proof.* Springer-Verlag, Berlin, Heidelberg, 333. https://doi.org/10.1007/978-3-540-87827-8_28
[6] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. 2013. A Machine-Checked Proof of the Odd Order Theorem. In *Proceedings of the 4th International Conference on Interactive Theorem Proving* (Rennes, France) *(ITP'13).* Springer-Verlag, Berlin, Heidelberg, 163–179. https://doi.org/10.1007/978-3-642-39634-2_14
[7] Michael Huth and Mark Ryan. 2008. *Lógica em Ciência da Computação* (2 ed.). LTC, Rio de Janeiro.
[8] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2010. SeL4: Formal Verification of an Operating-System Kernel. *Commun. ACM* 53, 6 (jun 2010), 107–115. https://doi.org/10.1145/1743546.1743574
[9] Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115. http://xavierleroy.org/publi/compcert-CACM.pdf
[10] Angel Mora, Emilio Muñoz-Velasco, and Joanna Golińska-Pilarek. 2011. Implementing a relational theorem prover for modal logic. *International Journal of Computer Mathematics* 88, 9 (2011), 1869–1884.
[11] Tobias Nipkow and Gerwin Klein. 2014. *Concrete Semantics: With Isabelle/HOL.* Springer Publishing Company, Incorporated.
[12] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2019. *A Proof Assistant for Higher-Order Logic.* Springer-Verlag. https://isabelle.in.tum.de/documentation.html Isabelle2019.
[13] Christine Paulin-Mohring. 2015. Introduction to the Calculus of Inductive Constructions. In *All about Proofs, Proofs for All*, Bruno Woltzenlogel Paleo and David Delahaye (Eds.). Studies in Logic (Mathematical logic and foundations), Vol. 55. College Publications. https://hal.inria.fr/hal-01094195

[2] In Coq, we can use sections to delimit the scope of local variables.