Formalizing Cut Elimination of Coalgebraic Logics in Coq[*]

Hendrik Tews

Institute of Systems Architecture, TU Dresden, Germany
http://askra.de/

Abstract. In their work on coalgebraic logics, Pattinson and Schröder prove soundness, completeness and cut elimination in a generic sequent calculus for propositional multi-modal logics [1]. The present paper reports on a formalization of Pattinson's and Schröder's work in the proof assistant Coq that provides machine-checked proofs for soundness, completeness and cut elimination of their calculus. The formalization exploits dependent types to obtain a very concise deep embedding for formulas and proofs. The work presented here can be used to verify cut elimination theorems for different modal logics with considerably less effort in the future.

1 Introduction

In [1], Pattinson and Schröder give two generic proofs of cut elimination for propositional multi-modal logics. In their framework a concrete modal logic is specified by a modal similarity type (i.e., a set of modal operators with arity) and a set of one-step rules. The semantics is given by a functor T together with a fibred predicate lifting for each modal operator. Models are T-coalgebras together with a valuation.

Pattinson and Schröder identify semantic conditions that allow them to prove soundness and cut-free completeness. Together, this gives the first cut-elimination theorem. They further identify purely syntactic conditions on the rule set that permit a syntactic cut-elimination proof. A formalization of these proofs in a proof assistant has many benefits beyond the mere validation of [1]. The formalization permits to obtain machine checked cut-elimination proofs for a variety of modal logics by verifying the semantic or syntactic preconditions only. Moreover, if the utilized proof assistant permits the extraction of executable code, then certified tautology checkers can be extracted from the completeness proof. Again, because of the modularity of Pattinson's and Schröder's work, the effort necessary for every new certified tautology checker will be relatively small. Finally, a cut-free calculus provides the foundation for a syntactic proof of Craig's interpolation property (and, indeed, [1] deals with Craig interpolation as an application). A formalization of cut-elimination therefore provides the basis

[*] This work was in part funded through the DFG project QuaOS.

for the verification of the interpolation theorem and for the extraction of certified programs that compute interpolants.

In this paper I describe the formalization of about $^2/_3$ of the results of [1] in the Coq proof assistant [2,3]. From now on, I refer to the specifications, theorems and proofs in Coq simply as *the formalization.* The formalization covers the generic syntax and semantics of coalgebraic logics, soundness, completeness, semantic and syntactic cut elimination. As an example, I use the modal logic K. The other material of [1], in particular Craig interpolation and the examples of coalition logic and the conditional logics CK and CK + ID are not (yet) contained in the formalization.

The size of the formalization is considerable. There are about 400 definitions and about 1300 theorems and lemmas, which are proved with more than $20,000$ lines of proof script in a total of $36,000$ lines of Coq code. About $6,000$ lines of Coq deal with standard results that are used but not proved in [1] (e.g., completeness and cut elimination for propositional logic). With several years of experience with the proof assistant PVS [4], I missed a few convenient features of the PVS user interface during the work on the formalization. This led to the implementation of automatic library compilation in Proof General [5, Sect. 11.2] and the proof-tree visualization program Prooftree [6]. The complete Coq sources of the formalization and some technical documentation are freely available on the internet [7].

A formalization of this extent does always uncover a number of typos and errors in the formalized work. It is a clear sign for the quality and accurateness of the pen-and-paper proofs of Pattinson and Schröder that I found only 4 errors beyond the level of nitpicking. The most serious one is probably that their substitution lemma 3.14 is wrong: The modal rank does not necessarily decrease as indicated. At first glance this seems to break the induction on modal rank in the completeness and in the cut-elimination theorem. However, with a suitably adapted substitution lemma, these proofs only need minor modifications. All errors that I describe here have been discussed with Dirk Pattinson and Lutz Schröder to confirm that these are indeed errors and not misunderstandings on my side.

Related work. Proof theory and, in particular, cut elimination is a very nice application for theorem proving. The formalization of cut elimination in a proof assistant provides an additional value, because cut-elimination proofs are complex and it is rarely ever the case that all cases are spelled out in pen-and-paper proofs. A long debate about the validity of a cut-elimination proof for the provability logic GL has only recently been resolved [8,9]. Depending on their aims, different authors use different approaches for their formalization of proof theory. Some use a shallow embedding of proofs and formalize only provability without an explicit representation of object-logic proofs (e.g., [10,11]). For cut elimination, one usually prefers a deep embedding of proofs, where object-logic proofs are terms and can be manipulated in the meta logic of the proof assistant (e.g., [9,12]). Formalizations in Isabelle/HOL that use a parametric rule set (e.g., [9,12]) need a well-formedness predicate on proof trees. The formalization presented here uses a deep embedding for formulas and proofs and a shallow embedding for models. The data types for formulas and proofs rely on dependent types to express the necessary side conditions in a very concise way without well-formedness predicates. The present work has some similarities with the work of Chapman [13] in that the formalization covers many different logics. In a sense, the scope of [13] is much broader, because it is not limited to propositional modal logics. However, while Chapman focuses on inversion, the present work proves soundness, completeness and cut elimination. Moreover, the framework of Chapman is not applicable to propositional modal logics, because its modal rules are, in general, not invertible.

Outline. This paper has a clear presentation problem. In order to be self-contained it should comprise the formalized material of [1], which is already at odds with the page limit. Describing several thousand lines of Coq specification and proofs at a level where the reader can follow the development is simply impossible. This paper must therefore focus on a few points of the formalization. Section 2 introduces a few aspects of Coq's logic and specification language. Section 3 presents in detail the deep embedding of coalgebraic logics that is used in the formalization. Section 4.3 high-lights interesting aspects of the formalization. In particular, this section discusses the differences between [1] and the formalization and the errors that I found. Section 5 gives an overview of the main results of the formalization. Section 6 concludes.

Up to the end of Section 3, this paper is self-contained. Section 4.3 and Section 5 can be read at a high level without particular knowledge of coalgebraic logics and without access to the source code of the formalization. However, for accurateness, I provide a few technical details in Section 4.3 that require familiarity with the proofs of [1]. Coq definitions that have been omitted for space reasons can be looked up in the documentation of the formalization [7].

Acknoledgements. I thank Dirk Pattinson and Lutz Schröder for several discussions on their paper.

2 Coq Preliminaries

In Coq, **Type** is a keyword that refers to one element in the infinite hierarchy of type universes in Coq. So A : **Type** simply means that A is a type. **Prop** is the type of propositions, which may or may not have a proof. Similar to higher-order logic, a set over A is conveniently modeled as a function $\mathsf{A} \to$ **Prop**.

In contrast to higher-order logic, Coq does not distinguish between types and terms. In Coq, there are only terms and every term has a type, which is a term again. Therefore, application is always written in the usual postfix way, even for terms that represent types at the conceptual level. For instance, as usual, $\mathsf{f\ a}$ stands for the application of function f to argument a. Using the same application, $\mathsf{list\ A}$ stands for the application of the type constructor list to type A, that is, for the lists over A, and $\mathsf{list\ (list\ A)}$ stands for the lists of lists over A.

In Coq, a propositions is a type whose inhabitants are its proofs. A proof for an implication F → G is a function that maps proofs of proposition F to proofs of proposition G. Consequently, the simple arrow denotes both, function types A → B and implications F → G.

Frequently used parameters can be declared as **Variable**'s in Coq. They are then automatically added to any definition in which they occur, saving the explicit declaration in each of them. The reader should understand a **Variable** as an arbitrary but fixed element of the given type. The Coq definitions and lemmas included in this paper do usually not mention declared variables. This is not always correct, but hopefully less confusing.

3 A Deep Embedding for Parametric Coalgebraic Logics

This section describes the base definitions for formulas, sequents, proof rules and proofs. The challenge in the formalization is that neither the formula syntax nor the rule set of the object logic is fixed, because the framework of [1] covers many different modal logics.

3.1 Formulas

Pattinson and Schröder take a simple propositional calculus with negation and conjunction and enrich it with modal formulas of the form $\heartsuit(A_1, \ldots, A_n)$, where $\heartsuit$ is a modal operator of arity n and the A_i are arbitrary formulas. The modal operators are drawn from a modal similarity type Λ. The whole development of [1] is parametric in Λ. In Coq, I define the type of Λ as a dependently typed record as follows.

```
Record modal_operators : Type :=
  { operator : Type; arity : operator → nat }.
```

The modal operators are given as an arbitrary type, the field arity determines their arity.

The following piece or source code shows the variable declarations for V, the set of propositional variables, and for L, the modal operators. They are used in almost all files of the formalization. Pattinson and Schröder assume the propositional variables to be a countably infinite set. This assumption will be explicitly added where needed.

```
Variable V : Type.
Variable L : modal_operators.

Inductive lambda_formula : Type :=
  | lf_prop : V → lambda_formula
  | lf_neg : lambda_formula → lambda_formula
  | lf_and : lambda_formula → lambda_formula → lambda_formula
  | lf_modal : forall(op : operator L),
      counted_list lambda_formula (arity L op) → lambda_formula.
```

The keyword **Inductive** introduces an inductive data type that is generated in the usual way from the given constructors. The type of formulas is called lambda_formula here and the constructors have an lf_ prefix, because Pattinson and Schröder use $\mathcal{F}(\Lambda)$ to denote it. The last constructor, lf_modal, for modal formulas, has a dependent type. It maps an operator op and a list of formulas to a new formula. In this paper, I write record selection as function application: operator L selects the type of operators and arity L op applies op to the arity function. The second argument of lf_modal must be a counted_list to ensure that its length matches the arity of op. For a type A and a natural number n, the type counted_list A n contains the lists over A of length n.[1] The use of dependent types is crucial here to capture the meaning of arity for modal operators.

3.2 Sequents

Pattinson and Schröder use a single-sided Gentzen-style sequent system. Sequents are defined as finite multisets of formulas. Multisets can be formalized as functions $A \longrightarrow \mathbb{N}$ or as a quotient type. In Coq both approaches have their drawbacks, because predicate extentionality, function extentionality as well as Hilbert's ϵ operator need additional axioms. I therefore decided to treat sequents as a setoid. A setoid is a type equipped with an equivalence relation, which represents the intended equality. As underlying type I simply use lists of formulas. Two such lists are equivalent, if one is a reordering or permutation of the other. In the formalization, I use the equivalence relation explicitly without relying on the Coq library of setoids.

```
Definition sequent : Type := list lambda_formula.

Inductive list_reorder(A : Type) : list A → list A → Prop :=
  | list_reorder_nil : list_reorder [] []
  | list_reorder_cons : forall(a : A)(l1 l2 : list A)(n : nat),
      list_reorder l1 l2 → list_reorder (a :: l1) ((firstn n l2) ++ a :: (skipn n l2)).
```

The equivalence relation on sequents is called list_reorder, because it is used for other types as well. It is defined here as an inductive relation on lists of an arbitrary type. The first constructor proves that the empty list is a reordering of itself. The second constructor proves that, whenever l1 is a reordering of l2, then also a :: l1 is a reordering of the list obtained by inserting a at an arbitrary position in l2. The functions firstn and skipn are from the Coq library. They return and cut off, respectively, the first n elements of a list; ++ denotes list concatenation.

The advantage of using lists of formulas as sequents is its simplicity. Many proofs can simply be done by induction on the list structure. The disadvantage is that the intended equality on sequents is not builtin: It always needs explicit treatment and, if forgotten, it may happen that a property holds for one sequent but not for some reordering of it.

[1] See [7] for the definition of counted_list and some other basic Coq material.

$$\frac{}{\vdash \Gamma, p, \neg p}\,(\mathrm{Ax}) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\,(\wedge) \qquad \frac{\vdash \Gamma, \neg A, \neg B}{\vdash \Gamma, \neg(A \wedge B)}\,(\neg\wedge)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \neg\neg A}\,(\neg\neg) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}\,(\mathrm{cut})$$

Fig. 1. Propositional rules

3.3 Rules and Rule Sets

A proof rule is a record with the assumptions and the conclusion.

```
Record sequent_rule : Type := {assumptions: list sequent; conclusion: sequent}.
```

Pattinson and Schröder do not distinguish between rules and rule instances: Proofs may only contain rules that appear literally in the respective rule set. For the propositional part of the calculus Pattinson and Schröder use the rule schemata in Figure 1. Because of the generice nature of [1], the modal rules of the calculus are not specified. Pattinson and Schröder only require that modal rules are *one-step rules*, see [1, Def. 3.3]. A one-step rule with k assumptions looks as follows:

$$\frac{\vdash a^1_1, \dots, a^1_{n_1},\ \neg b^1_1, \dots, \neg b^1_{m_1} \qquad \cdots \qquad \vdash a^k_1, \dots, a^k_{n_k},\ \neg b^k_1, \dots, \neg b^k_{m_k}}{\vdash \heartsuit_1(\dots), \heartsuit_2(\dots), \dots,\ \neg\heartsuit'_1(\dots), \neg\heartsuit'_1(\dots), \dots}$$

A rule of this form must fulfill 4 conditions in order to be a one-step rule: (1) all a^i_j and b^i_j must be variables, (2) the conclusion must not be the empty sequent, (3) all arguments of the modal operators in the conclusion must be (non-negated) variables and, finally, (4) all variables of the assumptions must appear in the conclusion.[2] In the framework of Pattinson and Schröder, a specific logic is specified by a set $\mathbf{R}$ of one-step rules, among others. Proofs may contain rules of the set $\mathcal{S}(\mathbf{R})$ of weakened substitution instances of $\mathbf{R}$. The set $\mathcal{S}(\mathbf{R})$ contains all rules $\Gamma_1\sigma\ \dots\ \Gamma_n\sigma\ /\ \Gamma_0\sigma, \Delta$ for a one-step rule $\Gamma_1 \dots \Gamma_k/\Gamma_0 \in \mathbf{R}$, an arbitrary substitution σ and an arbitrary weakening context Δ [1, Def. 3.5].

Rules are formalized as predicates on the type sequent_rule. For instance, for the $(\wedge)$-rule we have the following definition.

```
Definition is_and_rule(r : sequent_rule) : Prop :=
  exists(sl sr : sequent)(f1 f2 : lambda_formula),
    assumptions r = [sl ++ f1 :: sr; sl ++ f2 :: sr] ∧
    conclusion r = sl ++ (lf_and f1 f2) :: sr.
```

It is easy to see that is_and_rule is closed under sequent reordering in the following sense: Let s be the conclusion of a rule r, then, for every reordering s' of s there exists a rule r' such that s' is the conclusion of r'. This property is called rule_multiset in the formalization. It is proved for all rule sets and ensures that provability is closed under reordering.

[2] Condition (4) is missing in [1, Def. 3.3], see Section 4.3 below.

3.4 Proofs

To avoid confusion, one must distinguish between *meta-logic proofs* and *object-logic proofs*. The former are proofs in Coq (the meta logic), to establish properties of the latter. An object-logic proof is a proof in some coalgebraic logic.

Object-logic proofs are finite trees made of rule applications and hypotheses. Because Pattinson and Schröder frequently change the rule set and the hypotheses, I decided to make object-logic proofs parametric in the hypotheses and the rule set. Cut elimination is the main concern of the formalization. I therefore define object-logic proofs as a data type, whose elements can be manipulated by functions and meta-logic proofs. In the sense of [9] I use a deep embedding for derivations and rules (and variables).

```
Inductive proof(rules : set sequent_rule)(hypotheses : set sequent)
                                                      : sequent → Type :=
  | assume : forall(gamma : sequent),
       hypotheses gamma → proof rules hypotheses gamma
  | rule : forall(r : sequent_rule), rules r →
       dep_list sequent (proof rules hypotheses) (assumptions r) →
         proof rules hypotheses (conclusion r).
```

The type constructor for object-logic proofs takes three arguments: proof r h s is the type of proof trees with conclusion sequent s using rules and hypotheses from r and h, respectively. In a given proof tree, the sets of rules and hypothesis are constant throughout the tree (because these arguments are before the colon in the inductive definition). In contrast, the sequent may change: a proof tree of type proof r h (lf_and f g) typically contains subtrees of type proof r h f and proof r h g.

The constructor assume is for hypothesis leafs in the proof tree. It takes two arguments: the hypothesis gamma and a proof that gamma is indeed a member of the hypotheses. The constructor rule is for rule applications. It takes three arguments: a rule r, a proof that r is in the set of rules and a list of sub-proofs, one for each assumption of r. With all arguments present, it constructs a new proof tree for the conclusion of r.

The type dep_list of dependently typed lists, which occurs in the third argument of constructor rule, is slightly involved. Let T be a type constructor of arity one, A be a type and [a_1; a_2; ...; a_n] be a (conventional) list over A. Then, dep_list A T [a_1; a_2; ...; a_n] is a list of length n with the first element having type T a_1, the second having type T a_2, and so on until the last element of type T a_n. In the definition of object-logic proofs above, T is the partial application (proof rules hypothesis) that maps any sequent s to the type of proof trees with conclusion s. Therefore, dep_list sequent (proof rules hypotheses) (assumptions r) is the type of an inhomogeneous list that contains one proof for each assumption of the rule r.

3.5 Provability

Provability in the object logic is now straight-forward:

```
  Definition provable(rules : set sequent_rule)(hypotheses : set sequent)
                                                    (s : sequent) : Prop :=
    exists(p : proof rules hypotheses s), True.
```

Provability is not closed under reordering of the conclusion. This property is established as lemma.

```
  Lemma multiset_provability :
    forall(rules : set sequent_rule)(hypothesis : set sequent)(s r : sequent),
      rule_multiset rules →
      sequent_multiset hypothesis →
      list_reorder s r →
      provable rules hypothesis s →
        provable rules hypothesis r.
```

Here, sequent_multiset ensures that the set of hypothesis is closed under reordering and rule_multiset ensures the same for rules, as explained before.

It is worth noting again the succinctness of the deep embedding of coalgebraic logics in Coq. The definitions can be expressed in less than 20 lines and rely only on the type constructors list, dep_list and counted_list, where the first is from the Coq standard library and the other two are well-known under various names in the Coq literature. The dependent typing handles all side conditions. Separate predicates to ensure well-formedness of formulas and proofs are not necessary.

4 Highlights of the Formalization

This section presents some interesting aspects of the formalization, including the differences between [1] and the formalization and the few errors that the formalization revealed. Readers not familiar with [1] can safely skip over the technical details, which are only provided here for accurateness. Missing Coq definitions are described at a high level, readers interested in the source code are referred to [7].

4.1 Insufficient Intuitionistic Meta Logic

The object logic of Pattinson and Schröder is a classical logic and they also use classical logic in their reasoning. The logic of Coq is, however, intuitionistic. In Coq, neither $\neg\neg P \to P$ nor $P \vee \neg P$ can be proved in general. Obviously, one has to expect, that some results of Pattinson and Schröder are not provable in Coq. One can make Coq classical, by assuming, for instance, **forall**(P : **Prop**), ¬ (¬ P) → P as an axiom, which is available in a certain module of the standard library. Instead of the axiom, I prefer to use a property that must be explicitly listed in the assumptions of those results that depend on classical logic. The property, which is called classical_logic, is clearly visible in the sources and one can easily determine why theorems depend on it.

The points where classical reasoning is needed depends crucially on the encoding of sequents into formulas and on the fact that disjunction is encoded as negated conjunction in the object logic. The encoding of sequents into formulas is used for the semantics of sequents. Pattinson and Schröder associate the formula $\check{\Gamma} = \bigvee \Gamma$ with the sequent Γ and set $[\![\Gamma]\!] = [\![\check{\Gamma}]\!]$. Defining the finite disjunction $\bigvee \Gamma$ with an existential quantifier (i.e., $(A_1, \ldots, A_n)\check{} = \exists i \, . \, A_i$) is inappropriate, because the object logic does not contain quantification. I therefore use an iterative definition (i.e., $(A, \Gamma)\check{} = A \vee \check{\Gamma}$) which results in $(A_1, A_2)\check{} = \neg(\neg A_1 \wedge \neg A_2)$, because disjunction is syntactic sugar in the object logic.

For sequents with two or more formulas, the translation into formulas leads to some kind of Gödel-Gentzen double-negation translation. Therefore, a bit unexpected, the (Ax) rule can be proved sound without using classical_logic, because $\neg(\neg P \wedge \neg\neg P)$ is an intuitionistic tautology (contrary to $P \vee \neg P$).

In contrast, the soundness of the (cut) rule depends on classical_logic. Consider the case where Γ is the empty sequent and Δ contains one formula B. Then, soundness of (cut) amounts to $A \wedge \neg(\neg B \wedge \neg\neg A) \to B$, which is not an intuitionistic tautology, in contrast to $A \wedge (B \vee \neg A) \to B$.

The second point where classical_logic is needed in the formalization is the upward correctness of the ($\neg\neg$) rule, which is needed in the completeness proof. However, classical reasoning is here only required for the case where Γ is empty. For non-empty Γ the double-negation translation makes the statement provable.

Because of the effects of the double-negation translation, the soundness of the calculus with the (cut) rule and the completeness depend on classical_logic. Soundness without (cut) can be proved in intuitionistic logic.

There is only one third point in the whole formalization that requires classical_logic. This is a technical point inside Proposition 4.13, which is the base result for the completeness proof.

4.2 Differences in the Formalization

This subsection describes the important differences between the formalization and [1] and mentions some other noteworthy points. Errors and omissions that the formalization revealed are discussed in the next subsection.

Non-Negative Modal Rank. The modal rank of a formula or sequent is the maximal nesting level of modal operators in it. Purely propositional formulas have modal rank 0. Many proofs in [1] work by induction on the modal rank. Pattinson and Schröder occasionally use the modal rank -1 to avoid a case distinction, see for instance [1, Lem. 3.7]. No formula has rank -1.

For simplicity, the formalization uses natural numbers for the modal rank. To accommodate -1, the modal rank in the formalization is increased by one. That is, purely propositional formulas have rank 1, the formula $\heartsuit(p)$ has rank 2 for a propositional variable p, and so on. No formula has rank 0, but the empty sequent has rank 0.

Unused Results with Difficult Proofs. A few lemmas have been omitted from the formalization, mostly because no other results depend on them and they have a relatively difficult proof. One example is point 2 of Proposition 3.2, which recalls that the propositional rules including (cut) are complete with respect to propositional consequence. The proof of this result requires compactness, which is difficult to capture in the intuitionistic logic of Coq. The second example is the depth-preservation proof of Lemma 3.13, which is missing in the paper and which I discuss in the next subsection. The last example is Proposition 4.5 in [1] about *one-step completeness*. In coalgebraic logics, one-step completeness is a technical condition on the modal rules that implies completeness of the whole calculus. Proposition 4.5 states that it is sufficient to consider finite sets only for one-step completeness. This proposition is apparently only needed for the example of coalition logic.

Changes in the Syntactic Cut-Elimination Theorem. In [1], Proposition 5.6 for syntactic cut elimination states three properties together. First the admissibility of the non-atomic axiom rule, second the admissibility of contraction and third the admissibility of cut-elimination. Pattinson and Schröder prove all three properties in *one* mutual induction on the modal rank. In their proof, the induction step for non-atomic axioms of rank $n+1$ depends on cut elimination on rank n.

In the formalization I use two substitution lemmas (which are both derived from a more general result). One for the rule set including (cut) and one for the rule set without (cut). The latter one permits me to eliminate cut from the rule set before applying the substitution lemma. Then, the proof for non-atomic axioms in the formalization only requires cut elimination on purely propositional formulas of rank 0. Therefore, the result for non-atomic axioms is a separate proposition in the formalization, which is proved before the remainder of 5.6.

Injective Substitutions. Pattinson and Schröder use injective substitutions at two points in the syntactic cut-elimination proof, because injective substitutions preserve inclusion of multisets under certain conditions. (More accurately, $\Gamma \subseteq \Delta$ implies $\Gamma\sigma \subseteq \Delta\sigma$ for sequents Γ and Δ when σ is injective, Γ is a conclusion of a one-step rule and $\subseteq$ denotes inclusion on multisets.) For obtaining an injective substitution, they write, "*We may factorise $\sigma = \sigma_m \circ \sigma_e$ where σ_e is a renaming and σ_m is an injective substitution*" [1, page 29]. To avoid non-constructive definitions, I use a slightly weaker factorization. For a sequent Γ and a substitution σ I construct an injective σ'_m and a renaming σ'_e such that only $\Gamma\sigma = \Gamma\sigma'_e\sigma'_m$, while, in general, $\sigma \neq \sigma'_m \circ \sigma'_e$. Nevertheless, the cited sentence is one of the sentences with the biggest formalization overhead that I encountered. It required about 1500 lines of Coq and one week to construct σ'_m and σ'_e out of σ and to prove the necessary properties.

4.3 Omissions and Errors

In this subsection I discuss the non-trivial problems in the formal development of [1]. During the intense work on the formalization I also discovered a number of missing side conditions and obviously missing assumptions. These points are not included here. The fact that there are only 4 non-trivial problems in the proofs of [1] and that they have only negligible consequences for the main theorems, shows the accuracy of the pen and paper proofs of Pattinson and Schröder.

One Step Rules. The definition of one-step rules in [1] omits a side condition on the propositional variables: Just as described in Section 3.3, one must actually require that the assumptions do only use propositional variables that do appear in the conclusion. This condition is needed for those proofs that proceed by induction on the modal rank. For a substitution instance of a one-step rule, these proofs simply invoke the induction hypothesis on the assumptions of the rule. For this the modal rank of the assumptions must be smaller than the one of the conclusion. The simplest way to ensure this on substitution instances of one-step rules is the side condition on propositional variables.

In Coq, the fixed definition looks as follows:

```
Definition one_step_rule(r : sequent_rule) : Prop :=
  every_nth prop_sequent (assumptions r) ∧
  simple_modal_sequent (conclusion r) ∧
  conclusion r ≠ [] ∧
  every_nth
    (fun(s : sequent) ⇒
              incl (prop_var_sequent s) (prop_var_sequent (conclusion r)))
    (assumptions r).
```

The predicate every_nth P l is equivalent to Forall[3] from Coq's standard library, using a different and, for my purposes, more convenient definition. It expresses that P holds on all elements of the list l. The predicates prop_sequent and simple_modal_sequent express the constraints on the shape of the formulas in the assumptions and the conclusion, respectively. The predicate incl l1 l2 from the standard library holds if every element in l1 appears in l2 (regardless of multiplicity and order). The function prop_var_sequent : sequent $\to$ list V collects the propositional variables in a sequent.

Missing Proof for Depth Preservation. The inversion Lemmas 3.12 and 3.13 of [1] state that the inverted rules of $(\wedge)$, $(\neg\wedge)$ and $(\neg\neg)$ are *depth-preserving admissible.* This means, for instance, for the $(\wedge)$ rule, that, if $\Gamma, A \wedge B$ is provable, then so are Γ, A and Γ, B with proof trees of the same or smaller size. The Lemmas 3.12 and 3.13 differ in the rule set for which they make this statement. Lemma 3.12 makes the statement for proofs using the propositional rules only while Lemma 3.13 applies to proofs using propositional as well as modal rules.

[3] Note the case! Forall differs from the keyword **forall**.

The proof of Pattinson and Schröder for 3.13 uses their Lemma 3.9, which states an equivalence of proofs for the two different rule sets and relies then on 3.12. The problem here is that their Lemma 3.9 makes no statement about the size of the proof trees. So the proof of Pattinson and Schröder proves the inversion property, but not the depth-preserving part of the statement.

Depth preservation is important for the syntactic cut-elimination proof, because this proof uses induction on the size of the proof tree. However, in the syntactic cut-elimination proof only 3.12 is needed. Lemma 3.13 is (apparently) never used. In the formalization of Lemma 3.13 I only prove the inversion property and omit the depth-preservation part.

Fixed Substitution Lemma. The substitution lemma 3.14 of Pattinson and Schröder makes the following statement. Assume that Γ is provable with rules of modal rank at most n (implying that Γ has rank n) and that σ is a substitution that maps propositional variables to formulas of modal rank at most k (i.e., σ has rank k). Then $\Gamma\sigma$ is provable with rules of modal rank $n + k$, using additional assumptions from the set $\mathrm{Ax}_k = \{\neg A, A, \Delta \mid A$ a formula of rank k, Δ a sequent of rank $k\}$. The proof is very simple: One takes the same proof tree and substitutes a suitable element from Ax_k for every occurrence of the (Ax) rule. Consider for instance $\Gamma = \neg p, p, \heartsuit(p)$ of rank 1, which can directly be proved with the (Ax) rule, and the substitution σ of rank 1 that maps p to $\heartsuit(p)$. Then $\Gamma\sigma = \neg\heartsuit(p), \heartsuit(p), \heartsuit(\heartsuit(p))$ should match an assumption from Ax_1, which is impossible, because $\Gamma\sigma$ has rank 2.

The substitution lemma is used inside induction proofs on the modal rank for sequents Γ of rank 1 and substitutions σ of rank n. The idea is to reduce the modal rank $n + 1$ of $\Gamma\sigma$ to rank n of the elements of Ax_n, making it possible to apply the induction hypothesis to the elements of Ax_n. Therefore, the trivial change of permitting sequents of rank $n + k$ in the set Ax in the substitution lemma would fix the problem, but make the lemma useless.

For the formalization I define, for an arbitrary substitution σ

$$\mathrm{Ax}^n_\sigma \;=\; \{\neg p\sigma, p\sigma, \Delta \mid \; p \text{ a propositional variable, } \Delta \text{ a sequent of rank } n\}$$

In the substitution lemma, the proof of $\Gamma\sigma$ is permitted to use assumptions from Ax^{n+k}_σ, where n is the rank of Γ and k is the rank of σ, as before. In the proofs using the substitution lemma, one can apply the induction hypothesis on the two-element sequent $\neg p\sigma, p\sigma$, which has rank k only, and then use a suitable weakening lemma to obtain $\neg p\sigma, p\sigma, \Delta$.

The σ parameter in the set Ax^n_σ conveys some information through the application of the substitution lemma. This makes it possible to use the substitution lemma inside the proof of point 1 of Proposition 5.6 in [1], which states the admissibility of the non-atomic axiom rule. Pattinson and Schröder prove a special claim there by induction on the proof tree.

A Gap in the Completeness Proof. Proposition 4.13 in [1] states completeness for rank n, that is, if Γ of modal rank n is valid in the special n-step semantics, then it can be proved with rules of rank n. The proposition makes the statement actually twice, for the rule set including (cut) and, with stronger assumptions, for the rule set without (cut). We focus here on the proof for the rule set including the (cut) rule. The proof proceeds by induction on the modal rank of Γ. Inside the induction step the obligation to find a proof for Γ is made simpler by reducing the complexity of Γ step by step. In the first step, the propositional rules are applied until Γ has the form

$$\neg\heartsuit_1(\ldots),\ldots,\neg\heartsuit_k(\ldots),\ \heartsuit'_1(\ldots),\ldots,\heartsuit'_{k'}(\ldots),\ \neg q_1,\ldots,\neg q_m,\ q'_1,\ldots,q'_{m'} \qquad (*)$$

Pattinson and Schröder make now a case distinction: Either the left part with the modal formulas is valid or the right part with the propositional variables. In case of the right part one can simply use the (Ax) rule to construct the needed proof. In case of the left part one can use the one-step completeness of the rule set (which is an assumption of the proposition) and the induction hypothesis to obtain a proof for

$$\neg\heartsuit_1(\ldots),\ldots,\neg\heartsuit_k(\ldots),\ \heartsuit'_1(\ldots),\ldots,\heartsuit'_{k'}(\ldots) \qquad (\dagger)$$

The gap that remains in the proof of Pattinson and Schröder is how to obtain a proof of $(*)$ from a proof of $(\dagger)$. One obviously only needs a weakening lemma for the rule set including (cut). However, this result is missing from [1]. The weakening lemma 3.11 of [1] states weakening only for the rule set *without* (cut).

Weakening can be obtained with (cut) in a simple way, however, this requires the admissibility of non-atomic axioms. Non-atomic axioms are admissible, but this result is only proved much later in 5.6 and not available at this point. For the formalization I therefore proof the required weakening lemma by induction on the proof tree without using non-atomic axioms.

5 Main Theorems in the Formalization

This section presents the Coq source code of a few high-level theorems in the formalization. For space reasons, missing definitions must be looked up in the source code [7] if the explanations do not suffice.

The definitions and lemmas that deal with the semantics of coalgebraic logics need as third parameter a functor T, which is declared as **Variable**, similar to V and L.

```
Variable T : functor.
```

A functor is a record containing two functions, one for a mapping on types (the objects) and one for the mapping on functions (the morphisms). Additionally, functor contains proofs for the relevant properties, such as, for instance, the preservation of identity morphisms.

The first theorem shown is the completeness result without (cut).

```
Lemma cut_free_completeness :
  forall(enum_V : enumerator V)(LS : lambda_structure)
        (rules : set sequent_rule)(osr : one_step_rule_set rules)(s : sequent),
    classical_logic →
    non_trivial_functor T →
    one_step_cut_free_complete (enum_elem enum_V) LS rules osr →
    valid_all_models (enum_elem enum_V) LS s →
      provable (GR_set rules) empty_sequent_set s.
```

Here, enum_V is an enumerator (i.e., an injective function nat → V) for the variables. It is only needed for constructing substitutions inside the proof.[4] The lambda structure LS contains a predicate lifting of the functor T for each modal operator in the modal similarity type L together with the necessary properties. These predicate liftings are used for the semantics of the modal operators. The universally quantified variable osr is a proof for the fact that rules forms a set of one-step rules. This property appears as a quantified variable instead of as an assumption, because it is needed as argument of one_step_cut_free_complete, which expresses that the rule set rules is one-step cut-free complete with respect to the lambda structure LS. The predicate valid_all_models ensures that the sequent s is valid in all models, while non_trivial_functor T ensures that there is at least one such model. The term (enum_elem enum_V) produces one variable as witness that the set of variables is not empty. Both one_step_cut_free_complete and valid_all_models are only well-formed for non-empty variables sets V.

The next theorem is semantic cut elimination. Its proof first uses the soundness of the logic to derive the validity of those sequents that possess a proof. It then relies on cut_free_completeness to prove the existence of a cut-free proof.

```
Theorem semantic_admissible_cut :
  forall(enum_V : enumerator V)(LS : lambda_structure)
        (rules : set sequent_rule)(osr_prop : one_step_rule_set rules),
    classical_logic →
    non_trivial_functor T →
    one_step_sound (enum_elem enum_V) LS rules osr_prop →
    one_step_cut_free_complete (enum_elem enum_V) LS rules osr_prop →
      admissible_rule_set (GR_set rules) empty_sequent_set is_cut_rule.
```

Here, we have the additional assumption one_step_sound that ensures the one-step soundness and thereby soundness. The predicate is_cut_rule captures all instances of (cut) and admissible_rule_set R H C expresses that all rules in C are admissible for the rule set R and the assumptions H.

Finally, here is the syntactic cut elimination theorem. Syntactic cut elimination works by moving applications of the cut rule upwards in the proof until they finally disappear. The theorem depends on two additional **Variables**: a decidable equality relation on the operators and on the propositional variables.

```
Variables (op_eq : eq_type (operator L)) (v_eq : eq_type V).
```

[4] Actually, all proofs in the formalization only require finitely many distinct variables. The number of variables needed depends on the syntactic structure of the sequent s. Just like Pattinson and Schröder I simply assume infinitely many variables, because a suitable finite upper bound has not been identified yet.

```
Theorem syntactic_admissible_cut : forall(rules : set sequent_rule),
  countably_infinite V → one_step_rule_set rules →
  absorbs_congruence rules →
  absorbs_contraction op_eq v_eq rules →
  absorbs_cut op_eq v_eq rules →
    admissible_rule_set (GR_set rules) empty_sequent_set is_cut_rule.
```

This theorem also needs an enumerator for V (provided by countably_infinite) and the one-step property for rules, but here these points appear as conventional assumptions. The other assumptions are the three absorption properties, where the latter two need the decidable equalities.

Comparing the two cut elimination statements, we see that the syntactic one can be proved in intuitionistic logic and makes no assumptions on the functor T.

As an example, the formalization currently contains only the modal logic K (see e.g., [14]). Its purpose is to ensure that the general results of the formalization are applicable to a concrete logic and that all assumptions can be discharged as expected. For this example, natural numbers are used as propositional variables and the only modal operator $\Box$ is defined with an inductive date type. The example contains an application of each of the main results of the formalization. Here I only show syntactic cut elimination.

```
Theorem k_syntactic_cut :
  admissible_rule_set (GR_set k_rules) empty_sequent_set is_cut_rule.
```

This theorem uses the equivalent but non-standard rule set k_rules, which permits cut elimination, see [1, Ex. 4.6]. The theorem is proved with the theorem syntactic_admissible_cut and suitable lemmas for the absorption properties of K.

6 Conclusion and Future Work

This paper presents the formalization of about $2/3$ of [1] in the proof assistant Coq. The formalization contains the necessary definitions to formalize and prove the results on soundness, completeness and cut elimination of coalgebraic modal logics. The formalization contains the modal logic K as example, ensuring that definitions and theorems can be employed. Using this formalization, it should be possible to obtain machine checked cut-elimination proofs and certified tautology checkers for a number of different modal logics with relatively little effort.

There are many interesting directions for continuing the work presented here. First, it would be nice to cover more examples in order to obtain machine checked cut-elimination theorems for a number of different modal logics. Second, it would be interesting to also formalize the remainder of [1], in particular the results on the interpolation property. The third point are certified programs, for instance, for checking tautologies in a particular modal logic. From definitions and proofs, Coq can extract Haskell or OCaml programs, which are correct by construction. Because the completeness proof of Pattinson and Schröder is constructive, one should be able to obtain a tautology checker from it. In the current form of the formalization, program extraction does not work, because the completeness result is formulated as theorem only. For program extraction one must restructure the completeness result into the function that constructs the proof and a correctness proof of that function.

References

1. Pattinson, D., Schröder, L.: Cut elimination in coalgebraic logics. Information and Computation **208** (2010) 1447–1468
2. The Coq development team: The Coq proof assistant reference manual. LogiCal Project. (2012) Version 8.4.
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer Verlag (2004)
4. Owre, S., Rajan, S., Rushby, J., Shankar, N., Srivas, M.: PVS: Combining specification, proof checking, and model checking. In Alur, R., Henzinger, T., eds.: Computer Aided Verification. Volume 1102 of LNCS., Springer (1996) 411–414
5. Aspinall, D., Kleymann, T.: User Manual for Proof General 4.2. LFCS Edinburgh. (September 2012) Available at `http://proofgeneral.inf.ed.ac.uk`.
6. Tews, H.: Automatic library compilation and proof tree visualization for Coq Proof General. Presentation at the 3rd Coq Workshop, Nijmegen, 2011
7. Tews, H.: Formalized Cut Elimination of Coalgebraic Logics: Source Code and Documentation. TU Dresden. (April 2013) Available at `http://askra.de/science/coalgebraic-cut`.
8. Goré, R., Ramanayake, R.: Valentini's cut-elimination for provability logic resolved. In Areces, C., Goldblatt, R., eds.: Advances in Modal Logic, College Publications (2008) 67–86
9. Dawson, J.E., Goré, R.: Generic methods for formalising sequent calculi applied to provability logic. In Fermüller, C.G., Voronkov, A., eds.: Logic for Programming, Artificial Intelligence, and Reasoning; Proceedings of LPAR-17. Volume 6397 of LNCS., Springer (2010) 263–277
10. Doczkal, C., Smolka, G.: Constructive completeness for modal logic with transitive closure. In Hawblitzel, C., Miller, D., eds.: Certified Programs and Proofs - Second International Conference. Volume 7679 of LNCS., Springer (2012) 224–239
11. Chapman, P., McKinna, J., Urban, C.: Mechanising a Proof of Craig's Interpolation Theorem for Intuitionistic Logic in Nominal Isabelle. In Autexier, S. et. al., ed.: Proceedings of Intelligent Computer Mathematics, AISC and Calculemus 2008. Volume 5144 of LNCS., Springer (2008) 38–52
12. Dawson, J.E., Goré, R.: Formalised cut admissibility for display logic. In: 15th International Conference on Theorem Proving in Higher Order Logics, Proceedings. Volume 2410 of LNCS., Springer (2002) 131–147
13. Chapman, P.: Tools and techniques for formalising structural proof theory. PhD thesis, University of St Andrews (June 2010) available at `http://hdl.handle.net/10023/933`.
14. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2002)