Constructive Formalization of Classical Modal Logic

Christian Doczkal and Gert Smolka
Saarland University

May 16, 2011

This papers reports about the formalization of classical modal logic in the constructive type theory of the proof assistant Coq. We represent formulas and models and define satisfiability, validity, and equivalence of formulas. Our main results are a small model theorem and the computational decidability of satisfiability, validity, and equivalence. We consider a logic with modalities for one-step and reflexive transitive reachability.

1 Introduction

We are interested in the formalization of decidable logics in constructive type theory. Of particular interest are logics for reasoning about programs, as exemplified by PDL [4] and CTL [2]. Given that these logics enjoy the small model property, one would hope that they can be formalized in constructive type theory without using classical assumptions. In this paper, we report about the constructive formalization of K* [10], a sublogic of PDL and CTL that has modalities for one-step and reflexive transitive reachability. We employ the proof assistant Coq [12] with the Ssreflect extension [7].

Our formalization[1] represents formulas and models and defines an evaluation function and predicates that express satisfiability, validity, and equivalence of formulas. Our main results are a small model theorem and the computational decidability of satisfiability, validity, and equivalence.

We define computational decidability as follows:[2]

```
Definition decidable (X : Type) (P : X -> Prop) : Type := forall x : X, {P x} + {~ P x}.
```

Given that we work in a constructive type theory without assumptions, a proof of *decidable P* certainly ensures that P is computationally decidable. A proof of *decidable P* is a function that for each argument x yields a labeled proof of either Px or $\neg Px$. Note that every boolean predicate is decidable, and that every decidable predicate can be reflected into a boolean predicate (i.e., a function into *bool*). When we say decidable in the following, we mean decidability according to the above definition.

The logics we want to formalize come with a classical two-valued semantics. In particular, the evaluation function for formulas is two-valued. To account for this, we employ models whose

[1] The source code can be found at `http://www.ps.uni-saarland.de/~doczkal/coq-3/`
[2] Note that Coq's library Coq.Logic.Decidable comes with a weaker, proof-oriented notion of decidability.

relations are boolean predicates and define evaluation as a boolean predicate. Hence, viewed constructively, models are computational structures that come with a computable evaluation function. Viewed classically, our models assert the existence of certain functions that facilitate classical mathematical constructions. In particular, working with this kind of decidable models allows for the standard mathematical practice of defining functions from models to syntactic objects. We claim that our models give us a faithful representation of classical modal logic in constructive type theory.

Given that we work with computational structures and are interested in decidability results, Coq's Ssreflect extension turned out to be a good platform for our formalization. We make use of Ssreflect's support for boolean propositions, finite types, and finite sets [6, 5].

We could not find a formalization of classical modal logic in constructive type theory in the literature. However, there is work on the verification of decision procedures in constructive logic, often with an emphasis on code extraction. An early example is a decision procedure for classical propositional logic verified by Caldwell [1] in the Nuprl system. Another example is Buchberger's algorithm for polynomial rings verified by Théry [13] in Coq.

The paper explains our work in three steps. First we consider propositional logic, then basic modal logic, and finally K*. For each logic, we first formulate the mathematical theory we have formalized and then comment on its realization in Coq. The mathematical theory is formulated such that is fits formalization. The decidability proofs employ syntactic models based on Hintikka sets [8]. We always give a function that for a formula yields a finite collection of syntactic models such that the formula is satisfiable if and only if it is satisfied by one of the syntactic models. For modal logic we rely on the development in [9]. The ideas of the decidability proof originated with the work of Fischer and Ladner [4], Pratt [11], and Emerson and Halpern [3]. The formalizations of the three logics appear in separate files accompanying the paper.

2 Propositional Logic

We start with the mathematical theory of classical propositional logic we have formalized. We call this theory P. Theory P is arranged such that it fits constructive formalization and scales to modal logic. The key result is the decidability of satisfiability, validity, and equivalence of formulas.

We assume an alphabet of names called **variables** and declare the letters p and q as ranging over variables. **Formulas** are defined by the grammar

$$s, t ::= p \mid \neg s \mid s \land t \mid s \lor t$$

A **model** $\mathcal{M}$ is a set of variables. The **satisfaction relation** $\mathcal{M} \vDash s$ between models and formulas is defined by induction on formulas.

$$\mathcal{M} \vDash p \iff p \in \mathcal{M} \qquad \mathcal{M} \vDash s \land t \iff \mathcal{M} \vDash s \text{ and } \mathcal{M} \vDash t$$
$$\mathcal{M} \vDash \neg s \iff \text{not } \mathcal{M} \vDash s \qquad \mathcal{M} \vDash s \lor t \iff \mathcal{M} \vDash s \text{ or } \mathcal{M} \vDash t$$

Satisfiability, **validity**, and **equivalence** of formulas are defined as follows.

- s is **satisfiable** if $\mathcal{M} \vDash s$ for some model $\mathcal{M}$.
- s is **valid** if $\mathcal{M} \vDash s$ for all models $\mathcal{M}$.
- s and t are **equivalent** ($s \equiv t$) if $\mathcal{M} \vDash s$ iff $\mathcal{M} \vDash t$ for all models $\mathcal{M}$.

Proposition 2.1 Let s and t be formulas.

1. s is valid iff $\neg s$ is unsatisfiable.
2. $s \equiv t$ iff $(s \wedge t) \vee (\neg s \wedge \neg t)$ is valid.
3. If $s \equiv t$, then s is satisfiable iff t is satisfiable, and s is valid iff t is valid.

We define two functions η and $\sim$ from formulas to formulas by induction on formulas.

$$\begin{aligned} \eta p &= p & \sim p &= \neg p \\ \eta(\neg s) &= \sim s & \sim(\neg s) &= \eta s \\ \eta(s \wedge t) &= \eta s \wedge \eta t & \sim(s \wedge t) &= \sim s \vee \sim t \\ \eta(s \vee t) &= \eta s \vee \eta t & \sim(s \vee t) &= \sim s \wedge \sim t \end{aligned}$$

A formula s is **negation normal** if $\eta s = s$. The following propositions can be shown by induction on formulas.

Proposition 2.2 $\eta s \equiv s$ and $\sim s \equiv \neg s$.

Proposition 2.3 $\eta(\eta s) = \eta s$ and $\eta(\sim s) = \sim s$ and $\sim(\sim s) = \eta s$ and $\sim(\eta s) = \sim s$.

The **syntactic closure** Cs of a formula s is the set of all subformulas of s. We define Cs inductively.

$$\begin{aligned} Cp &= \{p\} & C(\neg s) &= \{\neg s\} \cup Cs \\ C(s \wedge t) &= \{s \wedge t\} \cup Cs \cup Ct & C(s \vee t) &= \{s \vee t\} \cup Cs \cup Ct \end{aligned}$$

A **Hintikka set** is a set H of negation normal formulas satisfying the following conditions:

1. If $\neg p \in H$, then $p \notin H$.
2. If $s \wedge t \in H$, then $s \in H$ and $t \in H$.
3. If $s \vee t \in H$, then $s \in H$ or $t \in H$.

Proposition 2.4 Let H be a Hintikka set. Then $\{\, p \mid p \in H \,\}$ is a model that satisfies every formula $s \in H$.

We now have a decision procedure for statisfiability. Given a formula s, the procedure checks whether the finite set $C(\eta s)$ has a subset that contains ηs and is a Hintikka set.

Theorem 2.5 A formula s is satisfiable if and only if there exists a Hintikka set $H \subseteq C(\eta s)$ such that $\eta s \in H$.

Proof By Propositions 2.1 and 2.2 we know that s is satisfiable iff ηs is satisfiable. Let $\mathcal{M} \vDash \eta s$. Then $\{\, t \in C(\eta s) \mid \mathcal{M} \vDash t \,\}$ is a Hintikka set containing ηs. The other direction follows with Proposition 2.4.

Corollary 2.6 Satisfiability, validity, and equivalence of formulas are decidable.

Proof Follows with Theorem 2.5 and Proposition 2.1.

3 Formalization of Propositional Logic

We now outline the formalization of P in Coq with Ssreflect. We start with the definition of types for variables, formulas, and models.

```
Definition var := nat.
Inductive form := Var : var -> form | ...
Definition model := var -> bool.
```

The satisfaction relation is obtained with a recursive evaluation function:

```
Fixpoint eval (M : model) (s : form) : bool := ...
```

The definitions of satisfiability, validity, and equivalence are straightforward.

```
Definition sat s : Prop := exists M, eval M s.
Definition valid s : Prop := forall M, eval M s.
Definition equiv s t : Prop := forall M, eval M s = eval M t.
```

The proof of Proposition 2.1 can be carried out constructively since formulas evaluate to booleans. For (1) the de Morgan law for the existential quantifier is needed, which is intuitionistically provable. An analogous proof of the statement

$$s \text{ satisfiable iff } \neg s \text{ is not valid}$$

is not possible at this point since it would require the de Morgan law for the universal quantifier, which is not provable intuitionistically. However, there is a straightforward proof of the above statement once we have proven that satisfiability is decidable. As is, we can prove that decidability of satisfiability implies decidability of validity and equivalence.

```
Definition decidable (X : Type) (p : X -> Prop) : Type := forall x : X, {p x} + {~ p x}.
Lemma dec_sat2valid : decidable sat -> decidable valid.
Lemma dec_valid2equiv : decidable valid -> forall s, decidable (equiv s).
```

The definition of the functions η and $\sim$ and the proof of the corresponding propositions is straightforward.

We define the syntactic closure operator C as a recursive function from formulas to lists of formulas.

```
Fixpoint synclos (s : form) : seq form := ...
```

Given a formula s, we obtain $C(\eta s)$ as a finite type F and identify the Hintikka sets over F by a boolean predicate:

```
Variable s : form.
Definition F : finType := seq_sub (synclos (eta s)).
Definition Hcond (t : F) (H : {set F}) :=
  match val t with
    | Neg (Var v) => ~~ (Var v \in' H)
    | And u u' => u \in' H && u' \in' H
    | Or u u' => u \in' H || u' \in' H
    | _ => true
  end.
Definition hintikka (H : {set F}) : bool := forallb t, (t \in H) ==> Hcond t H.
```

Here, \in' extends membership in {set F} from F to form, separating the definition of Hintikka sets and the membership proofs for synclos (eta s) associated with F.

To use seq_sub, we have to show that form is a countable type. We do this by embedding form into the type of finitely branching trees with labels from a countable type. This type is easily

shown countable and provides a generic construction that can be easily adapted to changes in the syntax.

We then prove Proposition 2.4 for Hintikka sets in {set F} and Theorem 2.5 for formulas in F.

```
Theorem decidability t : sat (val t) <-> existsb H, hintikka H && (t \in H).
```

From this, we obtain Corollary 2.6. For additional detail, we refer the interested reader to the theory files.

4 Modal Logic

We now present the mathematical theory of modal logic we have formalized. We call this theory K. We assume that the reader has seen modal logic before. We see the models of modal logic as transition systems where the states are labeled with variables. Formulas are evaluated at a state of a transition system. A primitive formula p holds at a state w if w is labeled with p, a formula $\Box s$ holds at w if s holds at all successors of w, and a formula $\Diamond s$ holds at w if s holds at some successor of w.

We assume an alphabet $\mathcal{V}$ of names called **variables** and declare the letters p and q as ranging over variables. **Formulas** are defined by the grammar

$$s, t ::= p \mid \neg p \mid s \wedge t \mid s \vee t \mid \Box s \mid \Diamond s$$

For simplicity we consider only negation normal formulas. A **model** $\mathcal{M}$ is a triple consisting of the following components:

- A **carrier** set $|\mathcal{M}|$ whose elements are called **states**.
- A relation $\to_{\mathcal{M}} \subseteq |\mathcal{M}| \times |\mathcal{M}|$ called **transition relation**.
- A function $\Lambda_{\mathcal{M}} \colon |\mathcal{M}| \to 2^{\mathcal{V}}$ called **labeling function**.

We deviate from the standard definition by admitting models with an empty set of states. This does not make a difference as it comes to satisfiability and validity of formulas. The **satisfaction relation** $\mathcal{M}, w \vDash s$ between models, states, and formulas is defined by induction on formulas.

$$\begin{aligned}
\mathcal{M}, w \vDash p &\iff p \in \Lambda_{\mathcal{M}} w & \qquad \mathcal{M}, w \vDash s \wedge t &\iff \mathcal{M}, w \vDash s \text{ and } \mathcal{M}, w \vDash t \\
\mathcal{M}, w \vDash \neg p &\iff p \notin \Lambda_{\mathcal{M}} w & \qquad \mathcal{M}, w \vDash s \vee t &\iff \mathcal{M}, w \vDash s \text{ or } \mathcal{M}, w \vDash t \\
\mathcal{M}, w \vDash \Box s &\iff \mathcal{M}, v \vDash s \text{ for all } v \text{ such that } w \to_{\mathcal{M}} v \\
\mathcal{M}, w \vDash \Diamond s &\iff \mathcal{M}, v \vDash s \text{ for some } v \text{ such that } w \to_{\mathcal{M}} v
\end{aligned}$$

Satisfiability, **validity**, and **equivalence** of formulas are defined as follows.

- s is **satisfiable** if $\mathcal{M}, w \vDash s$ for some model $\mathcal{M}$ and some state $w \in |\mathcal{M}|$.
- s is **valid** if $\mathcal{M}, w \vDash s$ for all models $\mathcal{M}$ and all states $w \in |\mathcal{M}|$.
- s and t are **equivalent** ($s \equiv t$) if $\mathcal{M}, w \vDash s$ iff $\mathcal{M}, w \vDash t$ for all models $\mathcal{M}$ and all states $w \in |\mathcal{M}|$.

We define a **negation operator** by induction on formulas.

$$\begin{aligned} \sim p &= \neg p & \sim(\neg p) &= p \\ \sim(s \wedge t) &= \sim s \vee \sim t & \sim(s \vee t) &= \sim s \wedge \sim t \\ \sim(\Box s) &= \Diamond(\sim s) & \sim(\Diamond s) &= \Box(\sim s) \end{aligned}$$

Proposition 4.1 Let s and t be formulas.

1. $\sim(\sim s) = s$
2. $\mathcal{M}, w \models \sim s$ iff not $\mathcal{M}, w \models s$
3. s is valid iff $\sim s$ is unsatisfiable.
4. s and t are equivalent iff $(s \wedge t) \vee (\sim s \wedge \sim t)$ is valid.

We define the **syntactic closure** Cs of a formula s by induction on s:

$$\begin{aligned} Cp &= \{p\} & C(\neg p) &= \{\neg p\} \\ C(s \wedge t) &= \{s \wedge t\} \cup Cs \cup Ct & C(s \vee t) &= \{s \vee t\} \cup Cs \cup Ct \\ C(\Box s) &= \{\Box s\} \cup Cs & C(\Diamond s) &= \{\Diamond s\} \cup Cs \end{aligned}$$

Note that Cs is a finite set consisting of subformulas of s.

A **Hintikka set** is a set H of formulas satisfying the following conditions:

1. If $\neg p \in H$, then $p \notin H$.
2. If $s \wedge t \in H$, then $s \in H$ and $t \in H$.
3. If $s \vee t \in H$, then $s \in H$ or $t \in H$.

A **Hintikka system** is a set of Hintikka sets. The **transition relation** $\to_S$ of a Hintikka system S is defined as follows: $H \to_S H'$ iff $H \in S$, $H' \in S$, and $t \in H'$ whenever $\Box t \in H$. We define the **model** $\mathcal{M}_S$ described by a Hintikka system S as follows: $|\mathcal{M}_S| = S$, $\to_{\mathcal{M}_S} = \to_S$, and $\Lambda_{\mathcal{M}_S} H = \{\, p \mid p \in H \,\}$. A **demo** is a Hintikka system $\mathcal{D}$ such that the following condition is satisfied: If $\Diamond s \in H \in \mathcal{D}$, then $H \to_{\mathcal{D}} H'$ and $s \in H'$ for some $H' \in \mathcal{D}$.

Proposition 4.2 Let $\mathcal{D}$ be a demo and $s \in H \in \mathcal{D}$. Then $\mathcal{M}_{\mathcal{D}}, H \models s$.

We now obtain a small model theorem.

Proposition 4.3 Let $\mathcal{M}$ be a model, s be a formula, and $H_w := \{\, t \in Cs \mid \mathcal{M}, w \models t \,\}$ for $w \in |\mathcal{M}|$. Then $\{\, H_w \mid w \in |\mathcal{M}| \,\}$ is a demo.

Proof Let $\mathcal{D} := \{\, H_w \mid w \in |\mathcal{M}| \,\}$. By induction on formulas it follows that the sets H_w are Hintikka sets. Moreover, $w \to_{\mathcal{M}} v$ always implies $H_w \to_{\mathcal{D}} H_v$. Now it is easy to see that $\mathcal{D}$ is a demo.

We now have a decision procedure for statisfiability.

Theorem 4.4 A formula s is satisfiable if and only if there exists a demo $\mathcal{D} \subseteq 2^{Cs}$ such that $s \in H$ for some $H \in \mathcal{D}$.

Proof Follows with Propositions 4.2 and 4.3.

Corollary 4.5 Satisfiability, validity, and equivalence of formulas are decidable.

Proof Follows with Theorem 4.4 and Proposition 4.1.

5 Formalization of Modal Logic

The most important design decision in the formalization of K is the representation of models. Since we are considering classical modal logic, formulas should evaluate to bool. From this it is clear that the transition relation and the labeling function should be formalized as boolean functions. Given the quantifications in the semantics of the modalities $\Box$ and $\Diamond$, we also require decidable quantification over the carrier. This leads to the following representation of models.

```
Record model : Type := Model {
  state : Type;
  trans : state -> state -> bool;
  label : state -> var -> bool;
  ex_dec : forall p : state -> bool, {ex p} + {~ ex p}
}.
```

Using ex_dec we define a boolean existential quantifier exs x, p x. We also obtain a boolean universal quantifier using the de Morgan law for existential quantification.

On the syntactic side we proceed similarly as we did for P. Given a formula s, we again represent the syntactic closure Cs as a finite type F. The definition of Hintikka sets is unchanged. The elements of {set {set F}} take the role of Hintikka systems. We define the transition relation TR and demos as follows.

```
Definition TR (S : {set {set F}}) (H H' : {set F}) : bool :=
  [&& H \in S, H' \in S & [set t | Box (val t) \in' H] \subset H'].

Definition Hdc (S : {set {set F}}) : bool :=
  forallb H, (H \in S) ==> forallb t, (t \in H) ==>
  if val t is Dia u then existsb H', TR S H H' && u \in' H' else true.

Definition demo (D : {set {set F}}) : bool := [&& Hdc D & D \subset [set H | hintikka H]].
```

Given these definitions, the results can be shown as one would expect from the mathematical proofs. Proposition 4.2 requires the construction of a finite model from a demo. Since Ssreflect provides boolean quantifiers for finite types, ex_dec is easily definable. Note that the definition of the finite set $\{H_w \mid w \in |\mathcal{M}|\}$ in Proposition 4.3

```
Definition H_at M (w : M) := [set t : F | w |= (val t)].
Definition D M := [set H | exs w : M, H == H_at w].
```

reveals that our construction of a demo from a model crucially depends on boolean quantification over states. Finally, we obtain Theorem 4.4.

```
Theorem decidability (t : F) :
  sat (val t) <-> existsb D, demo D && existsb H : {set F}, (t \in H) && (H \in D).
```

6 Modal Logic with Star Modalities

We now extend our theory K to modal logic with the star modalities $\Box^*$ and $\Diamond^*$. We call the extended theory K*. For K* we consider the formulas

$$s, t ::= p \mid \neg p \mid s \wedge t \mid s \vee t \mid \Box s \mid \Diamond s \mid \Box^* s \mid \Diamond^* s$$

and leave the definition of models unchanged. The definition of the satisfaction relation is extended by

$$\mathcal{M}, w \vDash \Box^* s \iff \mathcal{M}, v \vDash s \text{ for all } v \text{ such that } w \to_{\mathcal{M}}^* v$$
$$\mathcal{M}, w \vDash \Diamond^* s \iff \mathcal{M}, v \vDash s \text{ for some } v \text{ such that } w \to_{\mathcal{M}}^* v$$

where $\to_{\mathcal{M}}^*$ is the reflexive and transitive closure of $\to_{\mathcal{M}}$. Thus a formula $\Box^* s$ holds at a state w if all states reachable from w in $n \geq 0$ steps satisfy s, and a formula $\Diamond^* s$ holds at w if some state reachable from w in $n \geq 0$ steps satisfy s. The definitions of satisfiability, validity, and equivalence remain unchanged.

For the negation operator we need two additional equations:

$$\sim(\Diamond^* s) = \Box^*(\sim s) \qquad\qquad \sim(\Box^* s) = \Diamond^*(\sim s)$$

Now Proposition 4.1 can be shown for K*. Moreover, we have two important equivalences.

Proposition 6.1 $\Box^* s \equiv s \wedge \Box\Box^* s$ and $\Diamond^* s \equiv s \vee \Diamond\Diamond^* s$.

The equivalences motivate two additional conditions for Hintikka sets:

4. If $\Box^* s \in H$, then $s \in H$ and $\Box\Box^* s \in H$.
5. If $\Diamond^* s \in H$, then $s \in H$ or $\Diamond\Diamond^* s \in H$.

The syntactic closure operator is extended accordingly:

$$C(\Box^* s) = \{\Box^* s, \Box\Box^* s\} \cup Cs \qquad\qquad C(\Diamond^* s) = \{\Diamond^* s, \Diamond\Diamond^* s\} \cup Cs$$

Finally, the definition of demos is extended with the following condition:

- If $\Diamond^* s \in H \in \mathcal{D}$, then $H \to_{\mathcal{D}}^* H'$ and $s \in H'$ for some $H' \in \mathcal{D}$.

We need a property of the transition relation of Hintikka systems:

Proposition 6.2 Let S be a Hintikka system. If $H \to_S H'$ and $\Box^* s \in H$, then $\Box^* s \in H'$.

With this proposition we can show Proposition 4.2 for K*. It is not difficult to verify that Proposition 4.3, Theorem 4.4, and Corollary 4.5 carry over to K*.

Given the mathematical development of K* and the formalization of K, the formalization of K* is a matter of routine. To allow boolean evaluation of the star modalities, the models of K* must come with functions deciding the reflexive transitive closure of the transition relation.

```
Record model : Type := Model {
  ...
  trans_star_dec : forall w : state, decidable (clos_refl_trans trans w)
}.
```

The additional demo condition is easily expressed with Ssreflect's transitive closure operation connect for relations over finite types:

```
Definition Hrc (S : {set {set F}}) : bool :=
  forallb H, (H \in S) ==> forallb t, (t \in H) ==>
  if val t is (Dstar u) then existsb H', [&& connect (TR S) H H' , H' \in S & u \in' H']
  else true.
```

The decidability results then follow as for K.

6.1 Non-Compactness

Note that the star modalities yield a non-compact logic. Let $\Box^n s$ be defined by

$$\Box^0 s = s \qquad\qquad \Box^{n+1} s = \Box\Box^n s$$

and consider the set $\mathcal{A} = \{\Diamond^*\neg p\} \cup \{\Box^n p \mid n \in \mathbb{N}\}$. While $\mathcal{A}$ does not have a model, there exists a model for every finite subset of $\mathcal{A}$. Writing Box^n p for $\Box^n p$, we state this as follows:

```
Lemma unsat : ~ exists M : model, exists w : M,
                  w |= Dstar (Neg p) /\ forall n, w |= Box^n p.
Lemma sat_bound (n:nat) : exists M : model, exists w : M,
                  w |= Dstar (Neg p) /\ forall m, m <= n -> w |= Box^m p.
```

The first lemma is simple. We sketch the formal argument for the second lemma: For arbitrary n, we construct the model $\mathcal{M}$ where $|\mathcal{M}| = \{m \mid m < n + 2\}$, $\rightarrow_{\mathcal{M}}$ is the successor relation and all states except $n + 1$ are labeled with p. Clearly, $0 \vDash \Diamond^*\neg p$. To obtain $\forall m \leq n.\ 0 \vDash \Box^m p$, we show

$$\forall w \leq n.\ \forall m \leq n - w.\ w \vDash \Box^m p$$

by induction on m. Hence, $\mathcal{M}$ is a model as required. In Coq, $|\mathcal{M}|$ is represented using Ssreflect's finite ordinal types, so the definitions of ex_dec and trans_star_dec remain unchanged.

7 Conclusions

We have formalized an expressive modal logic in constructive type theory and proven a small model theorem and computational decidability. For this, we work without axioms and localize the required classical assumptions to the models. This gives us a straightforward definition of computational decidability.

Representing Models As noted in Section 5, the most important design decision in our formalization is the representation of models. A naive representation of models might look as follows:

```
  Record model : Type := Model {
  state : Type ;
  trans : state -> state -> Prop ;
  label : state -> var -> Prop
}.
```

In the constructive logic of Coq, these models are not faithful to classical modal logic, since they do not allow the definition of a two-valued evaluation function. The problem disappears if we assume informative excluded middle

```
Axiom IXM : forall P:Prop, { P } + { ~ P }
```

but then our definition of decidability no longer implies computational decidability. For this reason, we have localized specific instances of IXM to the models or, equivalently, required some relations to be boolean. Regarding the exact form of these instances, there is room for variation, provided that the following conditions are met:

1. The asserted decision functions must provide for a two-valued evaluation function satisfying the classical dualities.

2. The asserted functions need to be definable for finite carrier types.

Both conditions are essential for the results presented in this paper. Our formalization of models satisfies the above conditions. Below, we give an alternative formulation of models where we assume just enough functions to evaluate formulas to bool. For K, this amounts to

```
Record model := Model {
  state :> Type;
  trans : state −> state −> Prop;
  label : var −> (state −> bool);
  BOX : (state −> bool) −> (state −> bool);
  boxE : forall p w, BOX p w <−> forall v, trans w v −> p v
  DIA : (state −> bool) −> (state −> bool);
  diaE : forall p w, DIA p w <−> exists v, trans w v /\ p v;
}.
```

For this class of models, evaluation can be directly defined using the functions provided by the model.

```
Fixpoint eval M s :=
  match s with ... | Box s => BOX (eval M s) | Dia s => DIA (eval M s) end.
Notation "w |= s" := (eval s w).
```

Note that these models do not provide decidable quantification over their carrier. Our proof of the small model theorem relies crucially on decidable quantification. However, there is an alternative proof based on Pratt-style pruning [11, 9] that does not rely on decidable quantification.

Decidability and Choice If one assumes that in Coq's logic extended with excluded middle and choice our definition of decidability implies computational decidability, the naive representation of models is faithful to classical modal logic and can be used for proving decidability. The two axioms imply:

```
Lemma IXM_in_Prop : forall C:Prop, ((forall P:Prop, { P } + { ~ P }) −> C) −> C
```

While this does not allow the definition of a boolean evaluation function at the top level, it provides a proof of existence of such a function, which can be used for proving statements in Prop.

The above assumption is inspired by the elimination restriction Coq imposes on the sort Prop. This restriction ensures that functions into data types do not depend on proofs. However, we are not aware of any results, that would validate this assumption.

Constructive Metatheory We note that the constructive proofs of decidability we give here differ little from the mathematical arguments they are based on. This is mostly due to the fact that we chose to put rather strong assumptions into our models.

It appears that our constructive metatheory for classical logics does not extend to results like compactness. While it is possible to constructively prove decidability and non-compactness of K*, we are not aware of any constructive compactness argument for propositional logic.

K* is a sublogic of PDL and CTL. We are confident that our constructions carry over to these logics. More generally, a constructive formalization in the style of this paper should be possible for every logic with the small model property.

Acknowledgements

We thank Chad Brown for many valuable discussions about constructive type theory. We also thank the people from the Coq and Ssreflect mailing lists (in particular Georges Gonthier) for their helpful answers.

References

[1] James L. Caldwell. Classical propositional decidability via nuprl proof extraction. In Jim Grundy and Malcolm C. Newey, editors, *TPHOLs*, volume 1479 of *LNCS*, pages 105–122. Springer, 1998.

[2] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Programming*, 2(3):241–266, 1982.

[3] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. System Sci.*, 30(1):1–24, 1985.

[4] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. System Sci.*, pages 194–211, 1979.

[5] François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging mathematical structures. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *LNCS*, pages 327–342. Springer, 2009.

[6] Georges Gonthier, Assia Mahboubi, Laurence Rideau, Enrico Tassi, and Laurent Théry. A modular formalisation of finite group theory. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *LNCS*, pages 86–101. Springer, 2007.

[7] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA, 2008.

[8] K. Jaakko J. Hintikka. Form and content in quantification theory. Two papers on symbolic logic. *Acta Philosophica Fennica*, 8:7–55, 1955.

[9] Mark Kaminski, Thomas Schneider, and Gert Smolka. Correctness and worst-case optimality of pratt-style decision procedures for modal and hybrid logics. In Kai Brünnler and George Metcalfe, editors, *TABLEAUX 2011*, volume 6793 of *LNCS (LNAI)*, pages 196–210. Springer, Jul 2011. To appear.

[10] Mark Kaminski and Gert Smolka. Correctness of an incremental and worst-case optimal decision procedure for modal logic with eventualities. Technical report, Saarland University, Feb 2011.

[11] Vaughan R. Pratt. Models of program logics. In *Proc. 20th Annual Symp. on Foundations of Computer Science (FOCS'79)*, pages 115–122. IEEE Computer Society Press, 1979.

[12] The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 8.3 edition, 2010. `http://coq.inria.fr/`.

[13] Laurent Théry. A machine-checked implementation of Buchberger's algorithm. *J. Autom. Reasoning*, 26(2):107–137, 2001.