Constructive Formalization of Hybrid Logic with Eventualities

Christian Doczkal and Gert Smolka
Saarland University

This paper reports on the formalization of classical hybrid logic with eventualities in the constructive type theory of the proof assistant Coq. We represent formulas and models and define satisfiability, validity, and equivalence of formulas. The representation yields the classical equivalences and does not require axioms. Our main results are an algorithmic proof of a small model theorem and the computational decidability of satisfiability, validity, and equivalence of formulas. We present our work in three steps: propositional logic, modal logic, and finally hybrid logic.

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 [6] and CTL [4]. 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 H* [12], a hybrid logic [1] with eventualities (iteration in PDL, "exists finally" in CTL). We employ the proof assistant Coq [15] with the Ssreflect extension [9].

Our formalization represents formulas and models and defines a two-valued function evaluating formulas in models. Our main result is an algorithmic proof of a small model theorem, from which we obtain the computational decidability of satisfiability, validity, and equivalence of formulas. We do not require axioms and rely on the native notion of computability that comes with constructive type theory.

Hybrid logics [1] extend modal logics with nominals. The models of a modal logic can be seen as transition systems. The formulas of a modal logic describe

predicates on the states of a model. Nominals are primitive predicates that hold for exactly one state. Since we formalize a classical modal logic in constructive type theory, we require that the formulas denote boolean state predicates. To make this possible, we employ models that come with localized modal operations mapping boolean state predicates to boolean state predicates. While localized modal operations are essential to our constructive development, they are superfluous in a conventional mathematical development (since their existence is obvious). In the constructive setting, the localized modal operations constitute localized decidability assumptions, which eliminate the need for global decidability assumptions.

A conventional proof of a small model theorem starts from a formula and a model satisfying it. From the formula one obtains a finite syntactic closure (roughly the subformulas) and projects the states of the model to Hintikka sets contained in the closure. One then shows that the finitely many Hintikka sets obtained this way constitute a model of the formula.

The conventional proof does not work in our constructive setting since the Hintikka projection cannot be obtained from the model. However, there is an algorithmic technique known as pruning that originated with Pratt [14] that obtains from the system of all Hintikka sets contained in the finite syntactic closure a subsystem that constitutes a small model of all satisfiable formulas in the closure. As we show in this paper, the correctness of pruning can be shown constructively and provides for a constructive proof of the small model theorem. Interestingly, the pruning technique results in a worst-case optimal decision procedure (exponential complexity) while a naive search based on the small model theorem results in a double exponential decision procedure.

The formalization presented in this paper is based on the mathematical development presented in [11]. Small model theorems and pruning-based decision procedures originated in the work of Fischer and Ladner [6], Pratt [14], and Emerson and Halpern [5].

There appears to be no formalized decidability result for classical modal logic in the literature. Formalizing decidability results in classical logics like HOL requires an explicit formalization of computability. While there are formalizations of computability theory in HOL [13, 17], we are not aware of any decidability results based on these. 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 [3] in the Nuprl system. Another example is Buchberger's algorithm for polynomial rings verified by Théry [16] in Coq. Also, there is recent work on the constructive metatheory of classical logics. Ilik et. al. [10] give a constructive completeness proof for a classical first order sequent calculus with respect to

a certain class of Kripke models. In contrast to [10], we work with a notion of model that closely resembles the usual mathematical definition. We can do this since our proofs only require the construction of finite models.

Given that we work with finite structures and finite sets, we could profit much from Coq's Ssreflect extension. In particular, we make use of Ssreflect's support for boolean propositions, finite types, and finite sets [8, 7].

The paper presents our work in three cumulative steps: Propositional logic, modal logic with eventualities, and finally modal logic with eventualities and nominals. For each logic, we present the mathematical theory underlying the formalization and comment on its realization in Coq. In each case, we work with a finite formula closure and prove a small model theorem based on Hintikka sets. The Coq formalizations of the three logics appear in separate files that can be found at `http://www.ps.uni-saarland.de/extras/cpp11/`.

2 Propositional Logic

We start with the theory of classical propositional logic we have formalized. We call this theory P. Theory P is arranged such that it fits a constructive formalization that scales to modal logic. We first outline the mathematical theory and then sketch its formalization.

2.1 Mathematical Development

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

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

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.

$$\mathcal{M} \vDash p \iff p \in \mathcal{M} \qquad \mathcal{M} \vDash s \wedge t \iff \mathcal{M} \vDash s \text{ and } \mathcal{M} \vDash t$$
$$\mathcal{M} \vDash \neg p \iff p \notin \mathcal{M} \qquad \mathcal{M} \vDash s \vee t \iff \mathcal{M} \vDash s \text{ or } \mathcal{M} \vDash t$$

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 $\mathcal{M}$.

To express general negation we define a **negation operator** $\sim$ by induction on formulas:

$$\sim p = \neg p \qquad\qquad \sim(s \wedge t) = \sim s \vee \sim t$$
$$\sim(\neg p) = p \qquad\qquad \sim(s \vee t) = \sim s \wedge \sim t$$

Proposition 2.1 Let s and t be formulas.

1. $\sim\sim s = s$
2. $\mathcal{M} \vDash \sim s$ iff $\mathcal{M} \nvDash s$
3. s is valid iff $\sim s$ is unsatisfiable.
4. $s \equiv t$ iff $(s \wedge t) \vee (\sim s \wedge \sim t)$ is valid.

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

$$Cp = \{p\} \qquad\qquad C(\neg p) = \{\neg p\}$$
$$C(s \wedge t) = \{s \wedge t\} \cup Cs \cup Ct \qquad\qquad C(s \vee t) = \{s \vee t\} \cup Cs \cup Ct$$

We fix some formula s_0. A **Hintikka set** is a set $H \subseteq Cs_0$ satisfying:

H1. If $\neg p \in H$, then $p \notin H$.

H2. If $s \wedge t \in H$, then $s \in H$ and $t \in H$.

H3. If $s \vee t \in H$, then $s \in H$ or $t \in H$.

Proposition 2.2 Let H be a Hintikka set. Then $\{\, p \mid p \in H \,\}$ is a model that satisfies every formula $s \in H$.

Theorem 2.3 A formula $s \in Cs_0$ is satisfiable if and only if there exists a Hintikka set H such that $s \in H$.

Proof Let $\mathcal{M} \vDash s$. Then $\{\, t \in Cs_0 \mid \mathcal{M} \vDash t \,\}$ is a Hintikka set containing s. The other direction follows from Proposition 2.2.

We now have a decision procedure for satisfiability. Given a formula s, the procedure checks whether the finite set Cs has a subset that contains s and is a Hintikka set.

Corollary 2.4 Satisfiability, validity, and equivalence of formulas are decidable.

Proof Follows from Theorem 2.3 and Proposition 2.1.

2.2 Decidability, Finite Types and Finite Sets

We formalize our results in the proof assistant Coq, a system implementing the Calculus of Inductive Constructions [15]. All functions definable in Coq (without axioms) are total and computable. Hence, to show that a predicate `P : X -> Prop` is decidable we define a decision function of type

`forall x:X , { P x } + { ~ P x }`

returning for every `x:X` either a proof of `P x` or a proof of `~ P x`.

Our formal proofs rely heavily on the Ssreflect extension to Coq, so we briefly describe the most important features we use. For technical details refer to [7, 8].

Ssreflect defines an implicit coercion from `bool` to `Prop`, allowing booleans to appear in place of Propositions. The type of boolean predicates over a type `T` (i.e., `T -> bool`) is abbreviated `pred T`.

In Ssreflect, a finite type is a type together with an explicit enumeration of its elements. Finite types can be constructed from finite sequences using `seq_sub` and finiteness is preserved by many type constructors. For a sequence `xs:seq T` the finite type `X := seq_sub xs` comes with a generic injection `val` from `X` into `T`. Finite types come with boolean quantifiers `forallb` and `existsb` taking boolean predicates and returning booleans.

If `X` is a finite type, the type `{set X}` is the type of sets over `X`, which is itself a finite type. Ssreflect provides the usual set theoretic operations on `{set X}` including membership, written `x \in X`, and set comprehensions `[set x:X | p]`.

Ssreflect also provides a choice operator for boolean predicates over finite types. We use choice and boolean quantifiers to specify decision procedures in a declarative way.

2.3 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.

For convenience, we choose `nat` to be the type of variables. To obtain a representation that is faithful to classical logic, we represent models as boolean predicates. The satisfaction relation is then 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 (3) the de Morgan law for the existential quantifier is needed, which is intuitionistically provable. An analogous proof of the statement

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

is not possible at this point since it would require the de Morgan law for the universal quantifier, which is not provable intuitionistically. As is, we can prove that decidability of satisfiability implies decidability of validity and equivalence.

Lemma dec_sat2valid : decidable sat −> decidable valid.
Lemma dec_valid2equiv : decidable valid −> **forall** s, decidable (equiv s).

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_0, we obtain $C s_0$ as a finite type F.

Variable s0 : form.
Definition F : finType := seq_sub (synclos s0).

We identify Hintikka sets by a boolean predicate: [1]

Definition Hcond (t : F) (H : {set F}) :=
match val t **with**
  | NegVar v => ~~ (Var v \in' H)
  | And s t => s \in' H && t \in' H
  | Or s t => s \in' H || t \in' H
  | _ => true
end.
Definition hintikka (H : {set F}) : bool :=
  forallb t, (t \in H) ==> Hcond t H.

Our alternative membership \in' extends membership in {set F} from F to form, separating the definition of Hintikka sets and the membership proofs for synclos s0 associated with F. Defining Hintikka sets only for sets over F allows us to make use of Ssreflect's extensive library on finite sets.

We then prove Proposition 2.2 for Hintikka sets in {set F} and Theorem 2.3 for formulas in F.

Theorem decidability (t:F) :
  sat (val t) <−> **existsb** H, hintikka H && (t \in H).

From this, we obtain Corollary 2.4. See the theory file P.v for full details.

---

[1] The operators , &&, and ||, denote boolean negation, conjunction, and disjunction

3 Modal Logic

We now present the mathematical theory of modal logic with eventualities we have formalized. We call this theory K*. As before, we first outline the mathematical theory and then turn to formalization aspects.

3.1 Mathematical Development

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. A formula $\Box^* s$ ($\Diamond^* s$) holds at a state w if all (some) states reachable from w satisfy s. We call formulas of the form $\Diamond^* s$ **eventualities**.

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

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

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 $\Lambda_{\mathcal{M}} \colon \mathcal{V} \to 2^{|\mathcal{M}|}$ 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. We write $\to_{\mathcal{M}}^*$ for the reflexive transitive closure of $\to_{\mathcal{M}}$. The **satisfaction relation** $\mathcal{M}, w \vDash s$ between models, states, and formulas is defined by induction on formulas.

$$\mathcal{M}, w \vDash p \iff w \in \Lambda_{\mathcal{M}} p \qquad \mathcal{M}, w \vDash s \wedge t \iff \mathcal{M}, w \vDash s \text{ and } \mathcal{M}, w \vDash t$$

$$\mathcal{M}, w \vDash \neg p \iff w \notin \Lambda_{\mathcal{M}} p \qquad \mathcal{M}, w \vDash s \vee t \iff \mathcal{M}, w \vDash s \text{ or } \mathcal{M}, w \vDash t$$

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

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

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

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

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 $w \in |\mathcal{M}|$.

For a set of formulas A, we write $\mathcal{M} \vDash A$ if there exists some $w \in |\mathcal{M}|$ such that $\mathcal{M}, w \vDash t$ for all $t \in A$. We call a set of formulas A **satisfiable** if there is some model $\mathcal{M}$ such that $\mathcal{M} \vDash A$. We extend the **negation operator** to modal formulas:

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

Proposition 3.1 Let s and t be formulas.

1. $\sim(\sim s) = s$
2. $\mathcal{M}, w \vDash \sim s$ iff not $\mathcal{M}, w \vDash s$
3. s is valid iff $\sim s$ is unsatisfiable.
4. $s \equiv t$ iff $(s \wedge t) \vee (\sim s \wedge \sim t)$ is valid.
5. $\Box^* s \equiv s \wedge \Box\Box^* s$ and $\Diamond^* s \equiv s \vee \Diamond\Diamond^* s$.

We also extend the **syntactic closure**:

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

We again fix a formula s_0. A **Hintikka set** is a set $H \subseteq Cs_0$ satisfying (H1) to (H3) as defined for P and the following conditions (cf. Proposition 3.1(5)):

H4. If $\Box^* s \in H$, then $s \in H$ and $\Box\Box^* s \in H$.

H5. If $\Diamond^* s \in H$, then $s \in H$ or $\Diamond\Diamond^* s \in H$.

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 $\Lambda_{\mathcal{M}_S} p = \{H \in S \mid p \in H\}$. A **demo** is a Hintikka system $\mathcal{D}$ such that the following conditions are satisfied:

(D$\Diamond$) If $\Diamond s \in H \in \mathcal{D}$, then $H \to_{\mathcal{D}} H'$ and $s \in H'$ for some $H' \in \mathcal{D}$.

(D$\Diamond^*$) 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 3.2 Let $\mathcal{D}$ be a demo and $s \in H \in \mathcal{D}$. Then $\mathcal{M}_{\mathcal{D}}, H \vDash s$.

3.2 Demo Theorem

By Proposition 3.2, demos can be seen as syntactic models. We now show that every satisfiable formula $t \in Cs_0$ is satisfied by a demo. Note that, given s_0, there

are only finitely many demos. The **Hintikka universe** $\mathcal{H}$ is the (finite) set of all Hintikka sets. For models $\mathcal{M}$ and states $v \in |\mathcal{M}|$, we define $H_v := \{t \in Cs_0 \mid \mathcal{M}, v \vDash t\}$.

Proposition 3.3 Let $\mathcal{M}$ be a model and $v \in |\mathcal{M}|$. Then H_v is a Hintikka set.

Demos are closed under union. Hence, there exists a largest demo contained in $\mathcal{H}$. Starting from $\mathcal{H}$, we construct this demo by successively pruning sets that violate the demo conditions. The pruning technique originated with Pratt [14].

Proposition 3.4 Let S be a Hintikka system containing all satisfiable Hintikka sets. Then:

1. If $\Diamond t \in H \in S$ and $\forall H'. \, H \to_S H' \Rightarrow t \notin H'$, then H is unsatisfiable.
2. If $\Diamond^* t \in H \in S$ and $\forall H'. \, H \to_S^* H' \Rightarrow t \notin H'$, then H is unsatisfiable.

Proof 1. Assume $\mathcal{M}, w \vDash H$. Hence, there exists a state v such that $w \to_\mathcal{M} v$ and $\mathcal{M}, v \vDash t$. Thus, we have $t \in H_v$. This leads to a contradiction since $H \to_S H_v$. (H_v is satisfiable and therefore in S).

2. Assume $\mathcal{M}, w \vDash H$. Since H is a Hintikka set, we have $\Diamond \Diamond^* t \in H$. Hence, there exists a state v such that $\mathcal{M}, v \vDash \Diamond^* t$ and $H \to_S H_v$. To obtain a contradiction, it suffices to show that there exists a u such that $H_v \to_S^* H_u$ and $t \in H_u$. This follows easily by induction on $v \to_\mathcal{M}^* u$ and the fact that $v \to_\mathcal{M} u$ implies $H_v \to_S H_u$. ∎

We define a relation on Hintikka systems representing a single pruning action: $S \overset{p}{\to} S'$ iff $S' = S \setminus \{H\}$ for some H violating (D$\Diamond$) or (D$\Diamond^*$). We extend this to the **pruning relation** on Hintikka systems: $S \overset{p}{\rightsquigarrow} S'$ iff $S \overset{p}{\to}^* S'$ and S' is terminal for $\overset{p}{\to}$.

Proposition 3.5 Let S and S' be Hintikka systems such that $S \overset{p}{\rightsquigarrow} S'$. Then:
1. S' satisfies (D$\Diamond$) and (D$\Diamond^*$).
2. If S contains all satisfiable Hintikka sets, so does S'.

Let Δ be the set such that $\mathcal{H} \overset{p}{\rightsquigarrow} \Delta$. By Propositions 3.2 and 3.5, Δ is the demo containing exactly the satisfiable Hintikka sets and is thus uniquely determined.

Theorem 3.6 (Demo Theorem) A formula $t \in Cs_0$ is satisfiable if and only if there exists a Hintikka set $H \in \Delta$ such that $t \in H$.

Proof The direction form right to left follows from Proposition 3.2. For the other direction, assume $\mathcal{M}, v \vDash t$. Then $t \in H_v \in \Delta$.

We now have a decision procedure for satisfiability. Given an input formula s, the procedure constructs the set of all Hintikka sets contained in Cs. It then removes Hintikka sets violating (D◊) or (D◊*) until no such sets remain and returns satisfiable iff the resulting demo contains some H such that $s \in H$.

Corollary 3.7 Satisfiability, validity, and equivalence of formulas are decidable.

3.3 Formalization of Modal Logic

The most important design decision in formalizing modal logic is the representation of models. We require that formulas evaluate to boolean state predicates, i.e., functions of type state −> bool. To meet this requirement, we need boolean versions of the logical operations. For instance, for the ◊-modality we need an operation

  EXb : pred state −> pred state

satisfying

  forall p w : EXb p w <−> **exists** v, trans w v $\bigwedge$ p v

Since the boolean versions of the logical operations do not automatically exist in a constructive setting, we require that they are provided by the model. As it turns out, it suffices that a model comes with a boolean labeling function and the boolean operations for the existential modalities (i.e., ◊ and ◊*). This leads to the definition of models appearing in Fig. 1. The boolean operations for □ and □* can be defined from their duals EXb and EFb. For □* we have:

CoInductive AG X (R : X −> X −> Prop) (P : X −> Prop) (w : X) : Prop :=
  | AGs : P w −> (**forall** v, R w v −> AG R P v) −> AG R P w.
Definition AGb p w := ~~ EFb (fun v => ~~ p v) w.
Lemma AXbP p w : AGb p q <−> AG trans p w.

Note that the (co)inductive definitions of AG and EF are provably equivalent to more conventional definitions employing the reflexive transitive closure of the transition relation.

    We can now define a boolean evaluation function:

Fixpoint eval M s :=
  match s **with** Var v => label v |...| Dia s => EXb (eval M s) |... **end**.

We have now arrived at a faithful representation of classical modal logic providing the usual equivalences between formulas.

    On the syntactic side we proceed similarly as we did for P. Given a formula s_0, we again represent the syntactic closure Cs_0 as a finite type F. The definition of Hintikka sets is adapted to cover conditions (H4) and (H5). Hintikka systems are

Definition EX X (R : X −> X −> Prop) (P : X −> Prop) (w : X) : Prop :=
  exists v, R w v ⋀ P v.
Inductive EF X (R : X −> X −> Prop) (P : X −> Prop) (w : X) : Prop :=
  | EF0 : P w −> EF R P w
  | EFs v : R w v −> EF R P v −> EF R P w.
Record model := Model {
  state :> Type;
  trans : state −> state −> Prop;
  label : var −> pred state;
  EXb : pred state −> pred state;
  EXbP p w : EXb p w <−> EX trans p w ;
  EFb : pred state −> pred state;
  EFbP p w : EFb p w <−> EF trans p w }.

Figure 1: Definition of modal models

represented as elements of {set {set F}}. The transition relation $\to_S$ and the demo conditions (D◊) and (D◊*) are easily expressed as boolean predicates.

   Proposition 3.2 and Proposition 3.4 can be shown as one would expect from the mathematical proofs. Proposition 3.2 requires the construction of a finite model from a demo. Since the carrier of the constructed model is finite, label, EXb, and EFb are easily defined using Ssreflect's fintype and fingraph libraries.

   To constructively prove the demo theorem, we require some implementation of the pruning relation $\overset{p}{\rightsquigarrow}$. For this, we define a function

   pick_dia : {set {set F}} −> option {set F}

selecting, if possible, in a Hintikka system S some $H \in S$ violating (D◊). Likewise, we define a function pick_dstar for (D◊*). Both functions are defined using the choice operator provided by Ssreflect. From this, it is easy to define a pruning function:

Definition step S := **if** pick_dia S **is** Some H **then** S :\ H **else**
  if pick_dstar S **is** Some H **then** S :\ H **else** S.
Function prune (S : {set {set F}}) {measure (fun S => #|S|) S}
  : {set {set F}} := **if** step S == S **then** S **else** prune (step S).

It is easy to show that the result of pruning satisfies (D◊) and (D◊*). To obtain Proposition 3.5, we have to show that the precondition of Proposition 3.4 is an invariant of the pruning algorithm.

Definition HU := [set H | hintikka H].
Definition invariant (S: {set {set F}}) :=
  S \subset HU ⋀ **forall** H, H \in HU −> satF H −> H \in S.
Lemma invariant_prune S : invariant S −> invariant (prune S).

Finally, we obtain:

```
Theorem demo_theorem (t : F) :
  sat (val t) <-> existsb H, (H \in Delta) && (t \in H).
```

4 Hybrid Logic

Hybrid logic [2] extends modal logic with special variables called nominals that must label exactly one state. We extend K* with nominals and call the resulting logic H*.

4.1 Mathematical Development

We assume a countable set $\mathcal{N}$ of nominals and let x and y range over $\mathcal{N}$. The grammar of formulas is extended accordingly:

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

We extend the definition of models with a **nominal labeling** $\Phi_{\mathcal{M}} : \mathcal{N} \to 2^{|\mathcal{M}|}$ and require $|\Phi_{\mathcal{M}} x| = 1$ for all x.

We extend the **syntactic closure** to cover nominals:

$$Cx = \{x\} \qquad\qquad C(\neg x) = \{\neg x, x\}$$

As before, we fix a formula s_0 and define Hintikka sets as subsets of Cs_0. The Hintikka condition for nominals is identical to the condition for variables.

Constructing models $\mathcal{M}_S$ from arbitrary Hintikka systems S, does not work for H*. To extend Proposition 3.2 to H*, we adapt the notion of demo. A **demo** is a nonempty Hintikka system satisfying (D$\Diamond$) and (D$\Diamond^*$) as well as

(Dx) For every nominal $x \in Cs_0$, there exists exactly one $H \in \mathcal{D}$ such that $x \in H$.

We define the **model $\mathcal{M}_{\mathcal{D}}$** described by a demo $\mathcal{D}$ as follows: $|\mathcal{M}_{\mathcal{D}}|$, $\to_{\mathcal{M}_{\mathcal{D}}}$, and $\Lambda_{\mathcal{M}}$ are defined as for $\mathcal{M}_S$; for $\Phi_{\mathcal{M}_{\mathcal{D}}}$ we choose some $H_0 \in \mathcal{D}$ and define

$$\Phi_{\mathcal{M}_{\mathcal{D}}} x = \begin{cases} \{H_0\} & x \notin Cs_0 \\ \{H \in \mathcal{D} \mid x \in H\} & \text{otherwise} \end{cases}$$

Due to condition (Dx), every nominal is mapped to a singleton and we obtain:

Proposition 4.1 If $\mathcal{D}$ is a demo and $t \in H \in \mathcal{D}$, then $\mathcal{M}_{\mathcal{D}}, H \vDash t$.

4.2 Demo Theorem for Hybrid Logic

We now show that every satisfiable formula $t \in Cs_0$ is satisfied by a demo. We call a Hintikka system

- **nominally coherent** if it satisfies (Dx)
- **maximal**, if it is nominally coherent and contains all Hintikka sets not containing nominals.

Due to condition (Dx), demos for H^* are not closed under union. Hence, there is no largest demo and the pruning technique from Section 3.2 is not directly applicable. However, demos contained in a maximal Hintikka system are closed under union. This allows the search for a demo to be separated into two parts: guessing a suitable maximal Hintikka system and pruning it.

This two stage approach first appeared in [11], where it is used to obtain a complexity optimal decision procedure for hybrid PDL. In contrast to [11], where correctness is argued after establishing the small model property, we use the procedure as the basis for our algorithmic proof of the demo theorem.

Pruning a maximal Hintikka system may remove satisfiable Hintikka sets. To account for this, we refine the pruning invariant. Instead of requiring all satisfiable sets to be present, we state the invariant with respect to a model $\mathcal{M}$ and only require the sets H_w with $w \in |\mathcal{M}|$ to be present. We adapt Proposition 3.4 as follows:

Proposition 4.2 Let $\mathcal{M}$ be a model and S be a Hintikka system such that for all $w \in |\mathcal{M}|$, we have $H_w \in S$. Then:

1. If $\Diamond t \in H \in S$ and $\forall H'. H \rightarrow_S H' \Rightarrow t \notin H'$, then $\mathcal{M} \nVdash H$.
2. If $\Diamond^* t \in H \in S$ and $\forall H'. H \rightarrow_S^* H' \Rightarrow t \notin H'$, then $\mathcal{M} \nVdash H$.

We also need to adapt Proposition 3.5.

Proposition 4.3 Let $\mathcal{M}$ be a model and S be a maximal Hintikka system such that for all $w \in |\mathcal{M}|$, $H_w \in S$. If $S \overset{p}{\leadsto} S'$, then S' is nominally coherent and for all $w \in |\mathcal{M}|, H_w \in S'$.

Proposition 4.4 For every model $\mathcal{M}$, there exists a maximal Hintikka system S such that for all $w \in |\mathcal{M}|, H_w \in S$.

We fix a function Δ returning for a Hintikka system S some S' such that $S \overset{p}{\leadsto} S'$.

Theorem 4.5 A formula $t \in Cs_0$ is satisfiable iff there exists a maximal Hintikka system S such that $\Delta(S)$ is nominally coherent and contains some H such that $t \in H$.

Proof "⇒" Let $\mathcal{M}, w \vDash t$ and S be some maximal Hintikka system such that $H_w \in S$ for all $w \in |\mathcal{M}|$ (Proposition 4.4). Then $t \in H_w \in \Delta(S)$ and $\Delta(S)$ nominally coherent by Proposition 4.3.

"⇐" Satisfiability of t follows from Proposition 4.1, since $\Delta(S)$ is a demo by Proposition 3.5(1).

We now have a decision procedure for satisfiability. Given an input formula s, the procedure guesses for every nominal $x \in Cs$ a Hintikka set H such that $x \in H \subseteq Cs$. It then adds all Hintikka sets contained in Cs that do not contain nominals and prunes the resulting Hintikka system. It returns satisfiable iff the pruned Hintikka system contains for every $x \in Cs$ some H such that $x \in H$ and some H' such that $s \in H'$.

4.3 Formalization of Hybrid logic

To formalize H*, we first need to adapt the formal representation of models accordingly.

Record model := Model {
  ...
  nlabel : nvar −> pred state;
  nlabelP : **forall** x : nvar, **exists**! w, w \in nlabel x }.

This representation gives us all the required properties of nominals without having to assume that equality on state is decidable.

We define N to be the finite type of nominals occurring in F. We separate (Dx) into a nominal consistency condition Dxc requiring at most one occurrence of every nominal in N and a nominal existence condition Dxe requiring at least one occurrence. Condition Dxc is trivially preserved by pruning, while Dxe follows from the refined pruning invariant:

Definition invariant M (S : {set {set F }}) :=
  S \subset HU /\ **forall** v:M, H_at v \in S.
Lemma invariant_prune S : invariant S −> invariant (prune S).
Lemma invariant_xe S : invariant S −> Dxe S.

To prove Proposition 4.4 for a model $\mathcal{M}$, it is sufficient to prove the existence of a function assigning to every nominal in $x \in Cs_0$ the Hintikka set H_w, where $w \in |\mathcal{M}|$ is the unique w such that $\mathcal{M}, w \vDash x$

Lemma guess :
  exists f : N −> {set F}, **forall** x,
    exists2 w : M, eval M (val x) w & f x = H_at w.

This easily follows from the following choice principle

Lemma finite_choice (X : finType) Y (R : X −> Y −> Prop) :
  (**forall** x : X, **exists** y , R x y) −> **exists** f, **forall** x, R x (f x).

which is provable by induction on the enumeration of X. Finally we obtain:

Theorem demo_theorem (t : F) :
  sat (val t) <−> **existsb** S, maximal S &&
    let D := prune S in Dxe D && **existsb** H, (H \in D) && (t \in H).

Note that it is sufficient to check Dxe after pruning.

5 Conclusions

We have formalized propositional logic, modal logic with eventualities, and modal logic with eventualities and nominals in constructive type theory. Our main results are algorithmic proofs of small model theorems and the computational decidability of satisfability, validity, and equivalence of formulas.

    We represent models such that we can define a boolean evaluation function for formulas. This allows us to formalize classical modal logic. We do not assume axioms and employ the notion of computational decidability that comes with constructive type theory. This is possible since we localize the required classical assumptions to the models.

5.0.1 Representation of Models.

The most important design decision in our formalization is the representation of models. The reason for this is that in the constructive logic of Coq, the naive representation of models

Record naive_model : Type := Model {
  state : Type ;
  trans : state −> state −> Prop ;
  label : var −> state −> Prop }.

does not allow the definition of an evaluation function satisfying the classical equivalences of modal logic. This problem would disappear if we were to assume informative excluded middle

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

But then our definition of decidability would no longer imply computational decidability. Hence, we have localized specific instances of IXM to the models. [2] Regarding the exact form of these instances, there is room for variation, provided that the following conditions are met:

---

[2] EXb, EXbP, . . . are easily definable from IXM.

1. The class of models must admit an evaluation function for formulas satisfying the classical dualities.

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

We mention a second class of models for K*.

```
Record model := Model {
  state : Type;
  trans : state -> state -> bool;
  label : state -> var -> bool;
  exs : pred state -> bool;
  exsP p : exs p <-> exists w, p w ;
  trans_star : state -> state -> bool;
  trans_starP w v : trans_star w v <-> clos_refl_trans trans w v }.
```

For the purpose of this discussion, we call these models **strong models** and refer to the models defined in Section 3.3 as **weak models**.[3] The assumptions `exs` and `exsP` give us a decidable existential quantifier for states and boolean state predicates. This way one can define a boolean evaluation function directly following the mathematical presentation. The decidable existential quantifier also provides for a direct definition of a demo from a model:

Definition D (M:model) := [set H | exs (fun (w : M) => H == H_at w)]

This allows the formalization of the usual, non-algorithmic proof of the small model theorem.

Proposition 5.1 A formula $t \in Cs_0$ is satisfiable iff there exists a demo containing some H, such that $t \in H$.

Proof Let $\mathcal{M}, w \vDash t$. The set $\{ H_w \mid w \in |\mathcal{M}| \}$ is a demo as required. The other direction follows as before.

The algorithmic proof we have given for weak models provides a more informative small model theorem and shows that the additional strength of boolean existential quantification (i.e. `exs` and `exsP`) is not required to prove the small model theorem.

The file `Kstar_strong.v` contains a formal proof of a small model theorem for K* using the strong representation of models. The non-algorithmic formalization is not significantly shorter than the algorithmic formalization presented in Section 3.3.

---

[3] For every strong model, one can define a corresponding weak model. The converse does not seem to be true (consider a model $\mathcal{M}$ where $|\mathcal{M}| = \mathbb{N}$ and $n \rightarrow_{\mathcal{M}} m$ iff $n = m + 1$).

5.0.2 Extension to Temporal Logics.

The particular representation of models we use in this paper is motivated by the wish to find a design that extends, in a uniform way, to temporal logics like CTL [4]. Temporal logics employ models with a total transition relation and define the semantics of their modal operators using infinite paths. For the modal operators AF and EG one typically has the definitions

$$\mathcal{M}, w \vDash \mathsf{AF}\, s \iff \mathcal{M}, \sigma_n \vDash s \text{ for some } n, \text{ for all } \sigma \in \mathcal{M}^\omega \text{ such that } \sigma_0 = w$$

$$\mathcal{M}, w \vDash \mathsf{EG}\, s \iff \mathcal{M}, \sigma_n \vDash s \text{ for all } n, \text{ for some } \sigma \in \mathcal{M}^\omega \text{ such that } \sigma_0 = w$$

where $\mathcal{M}$ is a model, $\mathcal{M}^\omega$ is the set of all infinite paths in $\mathcal{M}$, and σ_n is the n-th state of an infinite path σ. The infinite path semantics does not seem to be feasible in constructive logic. However, inductive and coinductive definitions for AF and EG as we have used them in this paper for EF and AG seem to work fine:

Inductive AF (p : X −> Prop) (w : X) : Prop :=
| AF0 : p w −> AF p w
| AFs : (**forall** v, e w v −> AF p v) −> AF p w.
CoInductive EG (p : X −> Prop) (w : X) : Prop :=
| EGs v : p w −> e w v −> EG p v −> EG p w.

To support AF and EG, models would come with a boolean operator AFb and a proof AFbP that AFb agrees with AF for boolean predicates on the states of the model. With AFb and AFbP one can define EGb and a proof that EGb agrees with EG. With AFb and EGb one can then define an evaluation function satisfying the classical dualities. Moreover, given a finite type of states, one can define AFb and AFbP.

5.0.3 Acknowledgements.

References

[1] Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn et al. [2], pp. 821–868

[2] Blackburn, P., van Benthem, J., Wolter, F. (eds.): Handbook of Modal Logic, Studies in Logic and Practical Reasoning, vol. 3. Elsevier (2007)

[3] Caldwell, J.L.: Classical propositional decidability via nuprl proof extraction. In: Grundy, J., Newey, M.C. (eds.) TPHOLs. LNCS, vol. 1479, pp. 105–122. Springer (1998)

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

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

[6] Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. System Sci. pp. 194–211 (1979)

[7] Garillot, F., Gonthier, G., Mahboubi, A., Rideau, L.: Packaging mathematical structures. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs. LNCS, vol. 5674, pp. 327–342. Springer (2009)

[8] Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A modular formalisation of finite group theory. In: Schneider, K., Brandt, J. (eds.) TPHOLs. LNCS, vol. 4732, pp. 86–101. Springer (2007)

[9] Gonthier, G., Mahboubi, A., Tassi, E.: A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA (2008), `http://hal.inria.fr/inria-00258384/en/`

[10] Ilik, D., Lee, G., Herbelin, H.: Kripke models for classical logic. Ann. Pure Appl. Logic 161(11), 1367–1378 (2010)

[11] Kaminski, M., Schneider, T., Smolka, G.: Correctness and worst-case optimality of Pratt-style decision procedures for modal and hybrid logics. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNCS (LNAI), vol. 6793, pp. 196–210. Springer (2011)

[12] Kaminski, M., Smolka, G.: Terminating tableaux for hybrid logic with eventualities. In: Giesl, J., Hähnle, R. (eds.) IJCAR. LNCS, vol. 6173, pp. 240–254. Springer (2010)

[13] Norrish, M.: Mechanised computability theory. In: van Eekelen, M.C.J.D., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP. Lecture Notes in Computer Science, vol. 6898, pp. 297–311. Springer (2011)

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

[15] The Coq Development Team: The Coq Proof Assistant Reference Manual, 8.3 edn. (2010), `http://coq.inria.fr/`

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

[17] Zammit, V.: A mechanisation of computability theory in hol. In: von Wright, J., Grundy, J., Harrison, J. (eds.) TPHOLs. Lecture Notes in Computer Science, vol. 1125, pp. 431–446. Springer (1996)