Mechanizing Bisimulation Theorems for Relation-Changing Logics in Coq

Raul Fervari[1,2], Francisco Trucco[1] and Beta Ziliani[1,2]

[1]FaMAF, Universidad Nacional de Córdoba, Argentina
[2]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

Abstract. Over the last years, the study of logics that can modify a model while evaluating a formula has gained in interest. Motivated by many examples in practice such as hybrid logics, separation logics and dynamic epistemic logics, the ability of updating a model has been investigated from a more general point of view. In this work, we formalize and verify in the proof assistant Coq, the bisimulation theorems for a particular family of dynamic logics that can change the accessibility relation of a model. The benefits of this formalization are twofold. First, our results apply for a wide variety of dynamic logics. Second, we argue that this is the first step towards the development of a modal logic library in Coq, which allows us to mechanize many relevant results in modal logics.

1 Introduction

Historically, *modal logic* [10,11] has been understood as a logic to reason about different modes of truth. Under this perspective, it can be seen as an extension of propositional logic with *modalities*, that in certain contexts may have some particular interpretations, such as necessity, knowledge, belief, temporality, or obligation, just to name a few. Nowadays, *modal logics* is a term defining a family of logics to reason about relational structures, i.e., to reason about graphs. This is a consequence of the insights provided by the most common semantics for modal logics given in terms of the so-called Kripke structures [26]. In the wide spectrum of existing modal logics, one familiy has gained in interest over the last years: *dynamic modal logics*, i.e., logics that are able to update the model while evaluating the truth of a formula. Some well-known examples of this family are dynamic epistemic logics [37], separation logics [31,30], and hybrid logics [6].

The aforementioned examples of dynamic logics are specific instances designed with a particular goal in mind. Over the last years, there has been an increasing interest in understanding the behaviour of dynamic logics from a more general point of view (see e.g. [34,27,3]). Such perspective allows us to investigate the properties of abstract operators that are the building blocks used to construct concrete modalities, and to obtain a general perspective of the impact of including such kind of operators.

Some examples of results about operators that can change the accesibility relation of a model are collected in [18]. In [1,2] various abstract operations are presented, in particular, modalities to delete, add and swap-around an edge (both locally at an evaluation point and globally in the whole model). These logics are called *relation-changing logics*. In such works, particular notions of bisimulation for each operation are defined. A more general approach is taken in [3], where the notion of 'updating a relation' is generalized, and some results can be proved for all the logics encompassed in this framework. Also, the complexity of model checking is studied, whereas satisfiability is investigated in [4] (see also [27,8]). All these results require proofs that are tedious and of high complexity, so it would be interesting to benefit of the use of computational tools in order to guide or verify (parts of) the proofs.

The *Coq proof assistant* [9] is an interactive tool that helps us to perform complex mathematical proofs. It provides a formal language to formalize mathematical definitions, algorithms, theorems and their proofs. One of the main advantages of the Coq assistant, is that it allows us to build mathematical proofs constructively. The underlying logic in Coq is an intuitionistic logic with dependent types, known as *the calculus of inductive constructions (CIC)*. Thanks to the Curry-Howard correspondence, propositions are interpreted as types and proofs are interpreted as programs with the type of the corresponding proposition. Thus, we can say that Coq is essentially a type verifier (see e.g. [25]).

In the last years, several mathematical problems have been solved with the help of interactive tools like Coq; problems whose pen-and-paper proofs where put to doubt due to their complexity. For instance, in [22] a problem from graph theory known as *the four colour problem* was solved with the assistance of Coq. More recently, in [23] the *Kepler conjecture*, an open problem from combinatorial geometry, was proved by using a combination of the proof assistants HOL light [33] and Isabelle [29].

Motivated by examples as those in the last paragraph, we aim to develop a library to formalize and verify formal proofs in modal logic. In particular, we extend the formalization provided in [15] in order to model a particular family of dynamic logics called *relation-changing modal logics*. We introduce a family of dynamic operators, parameterized by a *model update function* that given a relational model, it returns a modified relational model. This follows the definitions introduced in [18,3]. Then we formalize a bisimulation notion which is agnostic with respect to the model update function, and mechanize the proof of the theorem of *invariance under bisimulation*. This theorem establishes that given two models that are related by a bisimulation, they satisfy the same formulas of the corresponding language. We consider this is the first step towards the development of a library for the mechanization of proofs for a wide variety of modal and dynamic logics.

Structure. In Sec. 2 we introduce the syntax and semantics of relation-changing modal logics, in which we have modalities parameterized by a model update function. In Sec. 3 we introduce the notion of bisimulation, and enunciate the invariance theorem. Then we focus on our main contribution: in Sec. 4 we present the formalization of these results in Coq. We conclude in Sec. 5 with some final remarks and future lines of research.

2 Basic Definitions

The syntax of the family of dynamic modal logics we call herein *relation-changing modal logics*, is a straightforward extension of the propositional logic. Let us introduce their syntax and semantics.

Definition 1 (Syntax). *Let* PROP *be a countable, infinite set of propositional symbols. The set* FORM *of formulas over* PROP *is defined as:*

$$\mathsf{FORM} ::= \bot \mid p \mid \varphi \to \psi \mid \blacklozenge_i \varphi,$$

where $p \in \mathsf{PROP}$, $\blacklozenge_i \in \mathsf{DYN}$ *a set of dynamic operators, and* $\varphi, \psi \in \mathsf{FORM}$. *Other Boolean operators are defined as usual.* $\blacksquare_i \varphi$ *is defined as* $\neg \blacklozenge_i \neg \varphi$.

For $S \subseteq \mathsf{DYN}$ *a set of dynamic operators, we call* $\mathcal{L}(S)$ *the extension of the propositional language allowing also the operators in* S. *If* S *is a singleton set* $S = \{\blacklozenge\}$, *we write* $\mathcal{L}(\blacklozenge)$ *instead of* $\mathcal{L}(\{\blacklozenge\})$.

Semantically, formulas of $\mathcal{L}(S)$ are evaluated in standard relational models.

Definition 2 (Models). *A model* $\mathcal{M}$ *is a triple* $\mathcal{M} = \langle W, R, V \rangle$, *where* W *is the* domain, *a non-empty set whose elements are called points or states;* $R \subseteq W \times W$ *is the* accessibility relation*; and* $V : \mathsf{PROP} \mapsto 2^W$ *is the* valuation.

Let w *be a state in* $\mathcal{M}$, *the pair* $(\mathcal{M}, w)$ *is called a pointed model; we usually drop parentheses and call* $\mathcal{M}, w$ *a pointed model.*

In this article, we restrict ourselves to models with only one accessibility relation (i.e., the underlying modal language has only one modal operator). A generalization to models with multiple accessibility relations is possible, but leads to further choices concerning the definition of the dynamic operators (e.g., which relation is affected by a given dynamic operator). Also, we only consider changes on the accessibility relation, but changes in the valuation would be easily incorporated in this framework.

Definition 3 (Model update functions). *Given a domain* W, *a* model update function for W *is a function* $f_W : W \times 2^{W^2} \to 2^{W \times 2^{W^2}}$, *that takes a state in* W *and a binary relation over* W *and returns a set of possible updates to the state of evaluation and accessibility relation.*

Let $\mathcal{C}$ *be a class of models, a* family of model update functions f *is a class of model update functions, one for each domain of a model in* $\mathcal{C}$*:*

$$f = \{f_W \mid \langle W, R, V \rangle \in \mathcal{C}\}.$$

$\mathcal{C}$ *is* closed under a family of model update functions f *if whenever* $\mathcal{M} = \langle W, R, V \rangle \in \mathcal{C}$, *then* $\{\langle W, R', V \rangle \mid f_W \in f, w \in W, (v, R') \in f_W(w, R)\} \subseteq \mathcal{C}$.

Clearly, the class of all pointed models is closed under any family of model update functions. In the rest of the article we only discuss the class of all models.

Notice, in the definition above, that a model update function is defined relative to a domain. We specifically require that all models with the same domain have the same model update function. This constraint limits the number of operators that can be captured in the framework, but at the same time leads to operators with a more uniform behavior. We will discuss this issue further after we introduce the formal semantics of the relation-changing operators below.

We now introduce the semantics for the general case.

Definition 4 (Semantics). *Let $\mathcal{C}$ be a class of models, $\mathcal{M} = \langle W, R, V\rangle$ be a model in $\mathcal{C}$, $w \in W$ a state, f a family of model update functions for $\mathcal{C}$ and $\blacklozenge_f$ its associated dynamic modality. Let φ be a formula in $\mathcal{L}(\blacklozenge_f)$. We say that $\mathcal{M}, w$* satisfies *φ, and write $\mathcal{M}, w \models \varphi$, when*

$$\begin{array}{lll}
\mathcal{M}, w \models \bot & & \textit{never} \\
\mathcal{M}, w \models p & \textit{iff} & w \in V(p) \\
\mathcal{M}, w \models \varphi \to \psi & \textit{iff} & \mathcal{M}, w \not\models \varphi \textit{ or } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \blacklozenge_f \varphi & \textit{iff} & \textit{for some } (v, R') \in f_W(w, R),\ \langle W, R', V\rangle, v \models \varphi.
\end{array}$$

The definition extends to languages with many modal dynamic operators in the obvious manner. A formula φ is satisfiable if for some pointed model $\mathcal{M}, w$ we have $\mathcal{M}, w \models \varphi$. We write $\mathcal{M}, w \equiv_{\mathcal{L}} \mathcal{N}, v$ when both models satisfy the same $\mathcal{L}$-formulas, i.e., for all $\varphi \in \mathcal{L}$, $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{N}, v \models \varphi$.

Notice, in the semantic definition, how the relation-changing modal operator $\blacklozenge_f$ potentially changes both the state of evaluation and the accessibility relation. On the other hand, the domain remains the same, and hence all $\blacklozenge_f$ operators in a formula are evaluated using the same model update function.

Examples of Relation-Changing Logics. First, notice that the classical modal diamond $\Diamond$ [10,11] is one particular instance of a dynamic operator, in which the accessibility relation remains unchanged and the evaluation state is changed by some successor via R. To simplify notation we use wv as a shorthand for $\{(w, v)\}$ or (w, v); context will always disambiguate the intended use. Let W a domain and $R \subseteq W^2$, the model update function associated to $\Diamond$ is defined as

$$f_W^{\Diamond}(w, R) = \{(v, R) \mid wv \in R\}.$$

Consider now the model update functions from [3]. Given a binary relation R, let us introduce some notation:

$$R^-_{wv} = R \backslash wv \qquad R^+_{wv} = R \cup wv \qquad R^*_{wv} = (R \backslash vw) \cup wv.$$

Define the following six model update functions, which give rise to natural dynamic operators: Van Benthem's sabotage operator $\blacklozenge_{\mathsf{gsb}}$ [34], and a local version $\blacklozenge_{\mathsf{sb}}$ that deletes an existing edge between the current state of evaluation and a successor state; a "bridge" operator $\blacklozenge_{\mathsf{gbr}}$ that adds an edge between two previously unconnected states, and a local version $\blacklozenge_{\mathsf{br}}$ that links the current state of evaluation and an inaccessible state; and the global and local versions ($\blacklozenge_{\mathsf{gsw}}$ and $\blacklozenge_{\mathsf{sw}}$, respectively) of an operation that swaps around edges of the model.

$$\begin{array}{ll}
f_W^{\mathsf{sb}}(w, R) = \{(v, R^-_{wv}) \mid wv \in R\} & \quad f_W^{\mathsf{gsb}}(w, R) = \{(w, R^-_{uv}) \mid uv \in R\} \\
f_W^{\mathsf{br}}(w, R) = \{(v, R^+_{wv}) \mid wv \notin R\} & \quad f_W^{\mathsf{gbr}}(w, R) = \{(w, R^+_{uv}) \mid uv \notin R\} \\
f_W^{\mathsf{sw}}(w, R) = \{(v, R^*_{vw}) \mid wv \in R\} & \quad f_W^{\mathsf{gsw}}(w, R) = \{(w, R^*_{vu}) \mid uv \in R\}.
\end{array}$$

It is easy to show that the basic modal logic $\mathcal{L}(\Diamond)$ enriched with other relation-changing modalities gains in expressivity. For example, the local sabotage operator $\blacklozenge_{\mathsf{sb}}$ and the local swap operator $\blacklozenge_{\mathsf{sw}}$ are logically stronger than the diamond operator when restricted to non-dynamic predicates, as the formulas $\blacklozenge_{\mathsf{sb}}p \rightarrow \Diamond p$ and $\blacklozenge_{\mathsf{sw}}p \rightarrow \Diamond p$ are valid. These operators are very expressive as they can force non-tree models (see e.g. [18,3]). For example, the formula $\blacksquare_{\mathsf{sb}}\Box\bot$ means that any local sabotage leads to a dead-end, hence the formula $\Diamond\Diamond\top \wedge \blacksquare_{\mathsf{sb}}\Box\bot$ can only be true at a reflexive state, a property that cannot be expressed in $\mathcal{L}(\Diamond)$.

3 Bisimulations

In modal model theory, the notion of bisimulation is a crucial tool. Typically, a bisimulation is a binary relation linking elements of the domains that have the same atomic information, and preserving the relational structure of the model. Because we need to keep track of the changes on the accessibility relation that the dynamic operators may introduce, bisimulations are defined as relations that link pairs of a state together with the current accessibility relation [18,3]. Notice that the notion we introduce is parameterized with a model update function, making the results general for the relation-changing logics from Sec. 2.

Definition 5 (Bisimulations). *Let $\mathcal{M} = \langle W, R, V\rangle$, $\mathcal{M}' = \langle W', R', V'\rangle$ be two models, and f a family of model update functions. A non empty relation $Z \subseteq (W \times 2^{W^2}) \times (W' \times 2^{W'^2})$ is an $\mathcal{L}(\blacklozenge_f)$-bisimulation if it satisfies the following conditions. If $(w, S)Z(w', S')$ then*

(atomic harmony) *for all $p \in \mathsf{PROP}$, $w \in V(p)$ iff $w' \in V'(p)$;*
(f-zig) *if $(v, T) \in f_W(w, S)$, there is $(v', T') \in f_{W'}(w', S')$ s.t. $(v, T)Z(v', T')$;*
(f-zag) *if $(v', T') \in f_{W'}(w', S')$, there is $(v, T) \in f_W(w, S)$ s.t. $(v, T)Z(v', T')$.*

Given two pointed models $\mathcal{M}, w$ and $\mathcal{M}', w'$ they are $\mathcal{L}(\blacklozenge_f)$-bisimilar (notation, $\mathcal{M}, w \,\underline{\leftrightarrow}_{\mathcal{L}(\blacklozenge_f)}\, \mathcal{M}', w'$) if there is an $\mathcal{L}(\blacklozenge_f)$-bisimulation Z such that $(w, R)Z(w', R')$ where R and R' are the respective relations of $\mathcal{M}$ and $\mathcal{M}'$.

Summing up, the bisimulation notion for each logic $\mathcal{L}(\blacklozenge_f)$ includes (atomic harmony) and the particular conditions for the model update function f. For instance, according to the above definition, the (zig) and (zag) conditions for the basic modal logic $\mathcal{L}(\Diamond)$ are defined as:

(zig) if $(w, v) \in S$, there is $v' \in W'$ s.t. $(w', v') \in S'$ and $(v, S)Z(v', S')$;
(zag) if $(w', v') \in S'$, there is $v \in W$ s.t. $(w, v) \in S$ and $(v, S)Z(v', S')$.

On the other hand, instantiating f with f^{sb} we get the following conditions:

(f^{sb}-zig) If $(w, v) \in S$, there is $v' \in W'$ s.t. $(w', v') \in S'$ and $(v, S^-_{wv})Z(v', S'^-_{w'v'})$;
(f^{sb}-zag) If $(w', v') \in S'$, there is $v \in W$ s.t. $(w, v) \in S$ and $(v, S^-_{wv})Z(v', S'^-_{w'v'})$.

In the same way, we can instantiate f with any of the model update functions mentioned in Sec. 2.

Theorem 1 (Invariance). *Let f be a family of model update functions, then $\mathcal{M}, w \underline{\leftrightarrow}_{\mathcal{L}(\blacklozenge_f)} \mathcal{M}', w'$ implies $\mathcal{M}, w \equiv_{\mathcal{L}(\blacklozenge_f)} \mathcal{M}', w'$.*

Proof. Let $\mathcal{M} = \langle W, R, V \rangle$ and $\mathcal{M}' = \langle W', R', V' \rangle$, such that $\mathcal{M}, w \underline{\leftrightarrow}_{\mathcal{L}(\blacklozenge_f)} \mathcal{M}', w'$. Then there exists Z such that $(w, R)Z(w', R')$.

We prove the theorem by structural induction. In fact, we prove a more general result. Let $S \subseteq W^2$, $S' \subseteq W'^2$ such that $(w, S)Z(w', S')$, we will show that $\langle W, S, V \rangle, w \equiv_{\mathcal{L}(\blacklozenge_f)} \langle W', S', V' \rangle, w'$. The base cases hold by (atomic harmony), and the $\to$ case is trivial.

[$\blacklozenge_f \varphi$ **case:**] Suppose $\langle W, S, V \rangle, w \models \blacklozenge_f \varphi$. Then there is $(v, T) \in f_W(w, S)$ s.t. $\langle W, T, V \rangle, v \models \varphi$. Because Z is a bisimulation, by (f-zig) we have $(v', T') \in f_{W'}(w', S')$ s.t. $(v, T)Z(v', T')$. By inductive hypothesis, $\langle W', T', V' \rangle, v' \models \varphi$ and by definition, $\langle W', S', V' \rangle, w' \models \blacklozenge_f \varphi$. For the other direction use (f-zag).

Therefore, since $(w, R)Z(w', R')$, we get $\mathcal{M}, w \equiv_{\mathcal{L}(\blacklozenge_f)} \mathcal{M}', w'$. □

Clearly the result holds when we extend $\mathcal{L}$ with any set of relation-changing modal operators. It suffices to require that the bisimulation comply with the various (f-zig) and (f-zag) conditions corresponding to all operators. In the next section we will reproduce the proof of Thm. 1 in Coq.

4 Formalization in Coq

4.1 The Coq Proof Assistant in a Nutshell

A typical proof in Coq looks like the following:

```
Lemma and_intro: forall (A B : Prop), A → B → A ∧ B.
Proof.
  intros A B HA HB. split.
  - apply HA.
  - apply HB.
Qed.
```

This simple proof states that if you are given a proof of proposition `A` and another proof of proposition `B`, then you have a proof for their conjunction. In order to be able to state the lemma and prove it, Coq presents three different domain specific languages: Gallina, The Vernacular and the tactics language Ltac. Gallina is Coq's mathematical higher-level language and program specification language. Seen as a programming language, Gallina is a dependently-typed functional language, while seen as a logical system, Gallina is a higher-order type theory. In the example, the lemma's statement (what follows the :) is written in this language.

The Vernacular allows the definition of functions or predicates, the statement of mathematical theorems and software specifications, the machine checking of proofs and the extraction of certified programs to different languages. In the example we use the following Vernacular *commands*: `Lemma` indicates the desire to state a theorem; `Proof` starts the proof; and `Qed` signals that the proof is completed, and therefore must be checked for errors and stored in the database of known facts if everything is correct. The reason for this check is to guarantee that the proof is indeed complete and that the tactics used to write the proof (see below) rightfully solved the problem.

Finally, the proof itself is written using the tactic language Ltac. In formal reasoning, a deduction rule is a link between a conclusion and a list of premises. There are two ways to understand a deduction rule. With *forward reasoning*, if we want to deduce the conclusion, we first try to deduce the list of premises and then use the deduction rule to prove the conclusion. With *backward reasoning* we go in the opposite direction: in order to prove the conclusion, we must prove the premises. Coq tactics are typically deduction rules that implement backward reasoning: when applied to a conclusion, usually called a *goal*, a tactic replaces this goal with the *subgoals* it generates, one for each premise of the rule. In the example, everything between `Proof` and `Qed` are tactic invocations. For instance, the tactic `split` replaces the goal `A` ∧ `B` with two subgoals, one for proving `A` and another for proving `B`.

Not all tactics are as simple as `split`. For instance, the tactic `tauto` implements a decision procedure for intuitionistic propositional calculus, so it is appropriate to solve many trivial statements (actually, it is capable of solving the above lemma in just one tactic invocation). Ltac also allows the definition of complex user defined tactics and decision procedures. See e.g. [9] for a more complete presentation of Coq.

4.2 Formalizing Relation-Changing Logics in Coq

In the rest of this section we will present our formalization of relation-changing modal logics in Coq, and the mechanization of the proof of the general invariance theorem (Thm. 1). The source code can be found at `http://tinyurl.com/rcml-in-coq0`. As we will see in the formalization below, it is easy to match the mathematical definitions from Secs. 2 and 3 with their counterpart in Coq.

Syntax. In order to formalize the syntax of relation-changing modal logics, we first need to define the countable set of propositional symbols PROP. We can accomplish this by using an inductive type definition:

```
Inductive prop : Set := p : nat → prop.
```

This creates a new type called `prop` with a type constructor `p` that given a natural number `n` constructs an inhabitant of the type `prop`, namely `p n`. It is clear that `prop` correctly formalizes the countable infinite set of propositions PROP.

Before we give the definition of the syntax we need to assume that a set of dynamic operators actually exists. This assumption is necessary because the definition of $\mathcal{L}(S)$, with $S \subseteq$ DYN, depends on the existence of a set of dynamic operators DYN.

```
Variable Dyn : Set.
```

Now we can give the definition of the set of formulae FORM, as in Def. 1:

```
Inductive form : Type :=
  | Atom    : prop → form
  | Bottom  : form
  | If      : form → form → form
  | DynDiam : Dyn → form → form.
```

Each line of this definition is interpreted as the members of the BNF from Def. 1. Other operators are defined as syntactic sugar. For example:

```
Definition DynBox (d : Dyn) (phi : form) : form := Not (DynDiam d (Not phi)).
```

Writing formulae with these constructors can be very tedious. Fortunately, Coq allows us to define a notation for them (we present the cases of dynamic operators; other operators are defined as expected):

```
Notation "<o> d phi" := (DynDiam d phi) (at level 65, right associativity).
```

Note that between parentheses we specify the precedence level and associativity.

Semantics. We now turn our attention to the formalization of the semantics of these logics. First, we need to formalize the concept of a relational model as those from Def. 2. In turn this requires to consider how powersets and relations are represented in Coq.

Let A be a set and `A` its correspoding formalization in Coq. In order to formalize a subset S of the power set 2^A, we can view S as a function that for each element $a \in A$ determines whether a belongs to S or not. Naively, one could think that S can be modeled with a function from `A` to `bool`. However, this is overly restrictive, as it will force us to make S decidable (Coq's functions are guaranteed to terminate). Thus, we use the constructive type `Prop` instead of `bool` and write `A → Prop` to mean "a subset of the 2^A".

Similarly, a binary relation R over A can be viewed as a function that given two elements $a, b \in A$ determines whether aRb or not. For this reason, we model R as `A → A → Prop`. Or, using Coq's standard library, simply `relation A`.

Now we are ready to introduce the formalization of the models of our logic. We can think a relational model as a triple consisting of a set `W`, a binary relation `R` defined over `W`, and a valuation function that for each element in `W` and each propositional symbol with type `prop`, decides whether that propositional symbol is valid or not in that element of `W`. In Coq we write the types of `W`, `R` and `V` as:

```
W : Set
R : relation W
V : W → prop → Prop
```

The type of the valuation function deserves an explanation. Above we define the valuation function as a function $V : \mathsf{PROP} \to 2^W$. Given a propositional symbol $p \in \mathsf{PROP}$, an element $w \in W$ and their respective formalizations `p : prop` and `w : W`, we formalize the expression $w \in V(p)$ as `V w p`.

Before we can give a definition of satisfiability we must give a formalization for the type of all model update functions. Remember that a model update function for a given domain W is a function $f_W : W \times 2^{W^2} \to 2^{W \times 2^{W^2}}$ that for each state of W and for each binary relation over W, associates a set of possible updates to the evaluation state and the accessibility relation. In Coq, we define the type of a model update function (`muf`) as:

```
Definition point (W: Set) : Type := (W * relation W).

Definition muf : Type := forall (W : Set),
  (point W) → (point W → Prop).
```

As with the mathematical definition f_W, a `muf` depends on the set W, and that is why we start with `forall (W : Set), ...` . For readability, we define a type `point W` to denote `(W * relation W)`, the Coq equivalent of $W \times 2^{W^2}$. Then, we define the `muf` as `(point W) → (point W → Prop)`. Note that the parentheses are just for readability: the function type → is right-associative.

Given that we have defined the notion of model update function and both the syntax and the models of our logic, we can now define the notion of satisfiability. It must be clear at this point that in order to define it, we need a function that assigns to each dynamic operator a model update function. We can simply assume that such function exists.

```
Variable F : Dyn → muf.

Fixpoint satisfies (W : Set) (R : relation W) (V : W → prop → Prop)
                   (w : W) (phi : form) : Prop :=
match phi with
| Atom a ⇒ V w a
| Bottom ⇒ False
| If phi1 phi2 ⇒ (satisfies W R V w phi1) → (satisfies W R V w phi2)
| DynDiam d phi ⇒
    let fw := F d W in
    exists (v : W) (R' : relation W), fw (w, R) (v, R') ∧ satisfies W R' V v phi
end.
```

Note how each case has one-to-one correspondence with Def. 4. For readability, we define the following notation:

```
Notation "# W , R , V >> w |= phi" := (satisfies W R V w phi) (at level 30).
```

Properties. With all of these definitions we can now formalize the notions of modal equivalence and bisimulation. For this part, we assume we work under the following assumptions:

```
Variables W W' : Set.
Variables (V : W → prop → Prop) (V' : W' → prop → Prop).
```

The notion of modal equivalence can be formalized unsurprisingly as:

```
Definition equivalent_at_points '(w, R) '(w', R') :=
  forall (phi:form), (# W , R , V >> w |= phi) ↔ (# W' , R' , V' >> w' |= phi).
```

(The notation `'(a, b)` just serves to say the definition takes a pair with components `a` and `b`.)

Before formalizing the notion of bisimulation for these logics, let us define the type of relations between models. As introduced in Def. 5, the type of relations defining bisimulations relates pairs of points and binary relations over the domains of the models:

```
Definition model_to_model_relation : Type :=
  (point W) → (point W') → Prop.
```

To formalize the notion of bisimulation, we first define each condition separately and then use them as functions in the definition of bisimulation (`f_zag` clause is analogous). To state these conditions, we work under the assumption that we have a relation between models `Z`:

```
Variable Z : model_to_model_relation.

Definition atomic_harmony : Prop :=
  forall w S w' S', Z (w, S) (w', S') → V w = V' w'.

Definition f_zig (f : muf) : Prop :=
  forall w S w' S' v T, Z (w, S) (w', S') →
    f W (w, S) (v, T) →
    (exists (v' : W') T', f W' (w', S') (v', T') ∧ Z (v, T) (v', T')).
```

Each condition shares the precondition of bisimulation (see Def. 5), namely that there is a relation `Z` between the models, and that the condition holds for every states `w` and `w'` and relations `S` and `S'` such that they are related according to `Z`.

Finally, the notion of bisimulation is defined as follows:

```
Definition bisimulation : Prop := atomic_harmony ∧
  (forall d : Dyn, (f_zig (F d))) ∧ (forall d : Dyn, (f_zag (F d))).

Definition bis_at_points (p: point W) (p': point W') : Prop :=
  bisimulation ∧ Z p p'.
```

Now we are ready to formally state the Theorem of Invariance under Bisimulation (Thm. 1):

```
Theorem InvarianceUnderBisimulation :
  forall (p: point W) (p': point W'),
  bis_at_points p p' → equivalent_at_points p p'.
```

Proof. The proof follows by structural induction on the formula `phi`. In order to get the right induction principle, we need to first massage the goal and the list of hypotheses a bit. The first lines of the proof are the following (we use Coq's `(* comments *)` to explain each line).

```
intros [w S] [w' S'].          (* name each component of the points *)
unfold bis_at_points.                     (* unfold definitions *)
unfold equivalent_at_points.
unfold bisimulation.
```

At this point, the goal looks like

```
(atomic_harmony ∧
 (forall d : Dyn, f_zig (F d)) ∧ (forall d : Dyn, f_zag (F d))) ∧
Z (w, S) (w', S') →
forall phi : form, # W, S, V >> w |= phi ↔ # W', S', V' >> w' |= phi
```

We split the different components of the first hypothesis, naming each of them again with the `intros` tactic. Then we introduce the formula `phi`.

```
intros [[HAtomicHarmony [HFZig HFZag]] HZwSw'S'].
intros phi.
```

Now our hypotheses looks like follows (the ... omits the trivial ones):

```
...
HAtomicHarmony : atomic_harmony
HFZig : forall d : Dyn, f_zig (F d)
HFZag : forall d : Dyn, f_zag (F d)
HZwSw'S' : Z (w, S) (w', S')
phi : form
```

With the goal being # `W, S, V` >> `w` |= `phi` ↔ # `W', S', V'` >> `w'` |= `phi`. We are almost ready to use structural induction on `phi`, but first we need to strengthen our inductive hypothesis, so we can use it on any points (i.e., on any pairs composed by a state and a binary relation). We do this by generalizing the goal:

```
generalize dependent S'. generalize dependent S.
generalize dependent w'. generalize dependent w.
```

The goal now looks like

```
forall (w : W) (w' : W') (S : relation W) (S' : relation W'),
Z (w, S) (w', S') →
# W, S, V >> w |= phi ↔ # W', S', V' >> w' |= phi
```

Now, we are ready to perform structural induction on `phi`:

```
induction phi as [p | | phi IHphi psi IHpsi | d phi IH];
```

The syntax above tells Coq how it should name the different hypothesis in the different cases. Notice that we usually end the tactics with a dot (.) but this time we ended the tactic `induction` with a semicolon (;). This tells Coq that the next tactic has to be applied to all cases in the induction. In particular we unfold the definition of `satisfies` and introduce all required variables and hypothesis.

```
simpl;               (* This tactic unfolds definitions *)
intros w w' S S' HZwSw'S'.
```

The atomic case is `V w p` ↔ `V' w' p`, which is solved with a simple rewrite using Atomic Harmony and a call to the tactic `tauto` mentioned in the previous section.

```
rewrite (HAtomicHarmony w S w' S' HZwSw'S').
tauto.          (* Solves the goal "V' w' p ↔ V' w' p" *)
```

For the bottom case we simply use `tauto`. Now for the if case, we split the proof into two separate directions. First we prove the left-to-right direction and then the right-to-left. This is performed using the `split` tactic. Luckily, we do not need to think about the two directions separately, since the same tactics work to prove both directions. That is why we end all tactics with a semicolon. Both directions are proved simply by assuming the two antecedents and then applying the inductive hypothesis of `psi` and `phi` and these two antecedents to the current subgoal.

```
split;
intros HIf Hsat;
apply (IHpsi w w' S S' HZwSw'S');
apply HIf;
apply (IHphi w w' S S' HZwSw'S');
apply Hsat.
```

The proof of the dynamic operator follows closely the same reasoning as the one present in Sec. 3. First, like with the if case, we use the tactic `split` to consider both directions, but this time we prove them separately. We will explain the left-to-right direction only, as the other direction is analogous. After `split` and `simpl` Coq tells us that we need to prove the following:

```
(exists (v : W) (R' : W → W → Prop),
   F d W (w, S) (v, R') ∧ # W, R', V >> v |= phi) →
exists (v : W') (R' : W' → W' → Prop),
  F d W' (w', S') (v, R') ∧ # W', R', V' >> v |= phi
```

First we give a name to the existentially quantified values of the antecedent, together with its properties:

```
intros [v [T [HfWwSvT HsatTv]]].
```

We get that `v : W`, that `T : relation W` and that they also satisfy two properties that we will name `HfWwSvT` and `HsatTv`:

```
HfWwSvT : F d W (w, S) (v, T)      (* (v,T) is a succesor via F d of (w,S) *)
HsatTv : # W, T, V >> v |= phi    (* the updated model satisfies phi *)
```

At this point we can apply the `HFZig` hypothesis in `HfWwSvT`:

```
apply (HFZig d w S w' S' v T HZwSw'S') in HfWwSvT
    as [v' [T' [HfW'w'S'v'T' HZvTv'T']]].
```

This introduces the following new hypotheses:

```
v' : W'
T' : relation W'
HfW'w'S'v'T' : F d W' (w', S') (v', T')
HZvTv'T' : Z (v, T) (v', T')
HsatTv : # W, T, V >> v |= phi
```

Remember that our goal at this point is to prove that:

```
exists (v'' : W') (R' : relation W'),
  F d W' (w', S') (v'', R') ∧ # W', R', V' >> v'' |= phi
```

The existential quantifiers are removed by providing witnesses: `v'` and `T'`:

```
exists v'. exists T'.
```

Now we only need to prove that:

```
F d W' (w', S') (v', T') ∧ # W', T', V' >> v' |= phi
```

The left proposition of the conjunction is identical to one of our hypothesis (namely `HfW'w'S'v'T'`). The other proposition can be proved by applying the inductive hypothesis to the `HsatTv` hypothesis. We can prove all this using the following tactics:

```
split.
* assumption.
* eapply IH. apply HZvTv'T'. assumption.
```

The proof is ended, so we issue the closing command `Qed`.

5 Final Remarks

In this work we formalized in the interactive proof assistant Coq, the syntax, semantics and a notion of bisimulation for a family of dynamic logics called *relation-changing logics*. These logics contain modalities that update the accessibility relation of the model while evaluating a formula. One particularity of our formalization is that, following the definition from [3], dynamic modalities are parameterized by a *model update function*, i.e., a function that given a pointed model, returns an updated pointed model. Thus, the definitions in Coq are simple, but powerful enough to encompass a whole family of logics. With these definitions at hand, we recreated the proof of the *invariance under bisimulation theorem:* if two models are bisimilar for a determined logic, then they satisfy the same formulas. Again, given the generality of our framework, we only need to prove one theorem, which holds for any instance of model update function (e.g., sabotage logics [34,32], swap logic [2], and others such as the program-based relation-changing operations from [21,35,19,20], which are as expressive as *propositional dynamic logic* [24] and whose bisimulation notion is the same as for the basic modal logic [10]). Moreover, the results we introduced can be straightforwardly extended to model update functions that also update the valuation of the model (see e.g. [36,7,5]).

There exist in the literature other works exploring the mechanization of proofs for modal and non-classical logics. In [15], a formalization of the basic modal logic $\mathcal{L}(\Diamond)$ is presented, in which we based our formalization. In addition, the author formalizes the extensions of the basic modal logic $S5$ and $S5^n$ [10], together with a natural deduction system. Then, the system is used to solve some logical puzzles. Regarding non-classical logics, in [39] the authors present a formalization in Coq of *linear logic*, together with the mechanization of some theorems for such logic, such as a proof of cut-elimination. A formalization in Coq of a Sahlqvists global correspondence theorem for the very simple Sahlqvist class is presented in [13]. From such formalization, it is possible to extract a verified Haskell program that computes correspondents of simple Sahlqvist formulas. Another approach has been taken in [38], in which a formalization and verification in Lean [14] of tableaux methods for modal logics is presented. Finally, recent works present a proof language for *differential dynamic logic* [12] for applications in cyber-physical systems, in the theorem prover KeYmaera X [28].

There are several interesting directions that we would like to explore in the future. The next step will be the mechanization of more complex results for relation-changing logics, such as the *Hennessy-Milner theorem* (i.e., the other direction of the proof presented here), and the correctness of the encodings into first-order and second-order logic [3]. Some of these results can be proved for particular instances of model updates functions, while others can be proved for the general case. Moreover, we would like to use our framework for mechanizing proofs for more concrete instances of this family, such as *dynamic epistemic logics* [37] and *modal separation logics* [16,17].

We consider that this work represents a first step towards a more serious use of an interative proof assistant for proofs in modal logic. One of the main problems in the aforementioned works, is that all the formalizations are too heterogeneous, and it makes difficult to reuse previous results. Our main goal is the development of a modal logic library that allows us the mechanization in a uniform way of a wide variety of results in modal and dynamic logics.

Ackowledgements. This work was partially supported by ANPCyT-PICTs-2017-1130 and 2016-0215, MinCyT Córdoba, SeCyT-UNC, and the Laboratoire International Associé INFINIS.

References

1. C. Areces, R. Fervari, and G. Hoffmann. Moving Arrows and Four Model Checking Results. In *WoLLIC 2012*, volume 7456 of *LNCS*, pages 142–153. Springer, 2012.
2. C. Areces, R. Fervari, and G. Hoffmann. Swap Logic. *Logic Journal of the IGPL*, 22(2):309–332, 2014.
3. C. Areces, R. Fervari, and G. Hoffmann. Relation-Changing Modal Operators. *Logic Journal of the IGPL*, 23(4):601–627, 2015.
4. C. Areces, R. Fervari, G. Hoffmann, and M. Martel. Satisfiability for relation-changing logics. *Journal of Logic and Computation*, 28(7):1443–1470, 2018.
5. C. Areces, D. Figueira, S. Figueira, and S. Mera. The Expressive Power of Memory Logics. *The Review of Symbolic Logic*, 4(2):290–318, 2011.
6. C. Areces and B. ten Cate. Hybrid Logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logic*, pages 821–868. Elsevier, 2007.
7. G. Aucher, P. Balbiani, L. Fariñas del Cerro, and A. Herzig. Global and local graph modifiers. *ENTCS*, 231:293–307, 2009.
8. G. Aucher, J. van Benthem, and D. Grossi. Modal logics of sabotage revisited. *JLC*, 28(2):269–303, 2018.
9. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*. Springer Publishing Company, Incorporated, 1st edition, 2010.
10. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.
11. P. Blackburn and J. van Benthem. Modal Logic: A Semantic Perspective. In *Handbook of Modal Logic*, pages 1–84. Elsevier, 2007.
12. B. Bohrer and A. Platzer. Toward structured proofs for dynamic logics. *CoRR*, abs/1908.05535, 2019.

13. C. D'Abrera and R. Goré. Verified synthesis of (very simple) sahlqvist correspondents via coq. In *AiML 2018, short presentations*, pages 26–30. College Publications, 2018.
14. L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The lean theorem prover (system description). In *CADE 2015*, pages 378–388, 2015.
15. P. de Wind. *Modal Logic in Coq.* Vrije Universiteit, 2001.
16. S. Demri and R. Fervari. On the complexity of modal separation logics. In *AiML 2018*, pages 179–198. College Publications, 2018.
17. S. Demri, R. Fervari, and A. Mansutti. Axiomatising logics with separating conjunction and modalities. In *JELIA 2019*, volume 11468 of *LNCS*, pages 692–708. Springer, 2019.
18. R. Fervari. *Relation-Changing Modal Logics.* PhD thesis, Universidad Nacional de Córdoba, Argentina, 2014.
19. R. Fervari and F. R. Velázquez-Quesada. Dynamic epistemic logics of introspection. In *DaLí 2017*, volume 10669 of *LNCS*, pages 82–97. Springer, 2017.
20. R. Fervari and F. R. Velzquez-Quesada. Introspection as an action in relational models. *JLAMP*, 108:1–23, 2019.
21. P. Girard, J. Seligman, and F. Liu. General dynamic dynamic logic. In *AiML 2012*, pages 239–260. College Publications, 2012.
22. G. Gonthier. Formal proof the four-color theorem. 2008.
23. T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. M. Rute, A. Solovyev, A. H. Thi Ta, T. N. Tran, D. T. Trieu, J. Urban, K. K. Vu, and R. Zumkeller. A formal proof of the kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.
24. D. Harel. *Dynamic Logic.* Foundations of Computing. The MIT Press, 2000.
25. W. A. Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.
26. S. Kripke. Semantical Analysis of Modal Logic I. Normal Propositional Calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.
27. Ch. Löding and P. Rohde. Model checking and satisfiability for sabotage modal logic. In P. Pandya and J. Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science: 23rd Conference, Mumbai, India, December 15-17, 2003. Proceedings*, pages 302–313, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
28. S. Mitsch and A. Platzer. The keymaera X proof IDE - concepts on usability in hybrid systems theorem proving. In *F-IDE@FM 2016*, volume 240 of *EPTCS*, pages 67–81, 2016.
29. L. C. Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.
30. D. Pym, J. Spring, and P.W. O'Hearn. Why separation logic works. *Philosophy & Technology*, 32(3):483–516, 2019.
31. J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.
32. P. Rohde. *On games and logics over dynamically changing structures.* PhD thesis, RWTH Aachen, 2006.
33. K. Slind and M. Norrish. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.
34. J. van Benthem. An Essay on Sabotage and Obstruction. In *Mechanizing Mathematical Reasoning*, pages 268–276, 2005.

35. J. van Benthem and F. Liu. Dynamic logic of preference upgrade. *Journal of Applied Non-Classical Logics*, 17(2):157–182, 2007.
36. H. van Ditmarsch, W. van der Hoek, and B. Kooi. Dynamic epistemic logic with assignment. In *AAMAS 2005*, pages 141–148. ACM, 2005.
37. H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Synthese Library. Springer, 2007.
38. M. Wu and R. Goré. Verified decision procedures for modal logics. In *ITP 2019*, volume 141 of *LIPIcs*, pages 31:1–31:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
39. B. Xavier, C. Olarte, G. Reis, and V. Nigam. Mechanizing focused linear logic in coq. *ENTCS*, 338:219–236, 2018.