SRI International

SRI-CSL Technical Report • November 2017, revised May 2022

PVS Embeddings of Propositional and Quantified Modal Logic

John Rushby
Computer Science Laboratory
SRI International, Menlo Park CA USA

John Rushby's research was partially supported by NASA Contract NNL16AA06B, Task Order NNL16AA96T, under a subcontract to Honeywell.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Abstract

Modal logics allow reasoning about various *modes* of truth: for example, what it means for something to be *possibly* true, or to *know* that something is true as opposed to merely *believing* it. This report describes embeddings of propositional and quantified modal logic in the PVS verification system. The resources of PVS allow this to be done in an attractive way that supports much of the standard syntax of modal logic, while providing effective automation.

The report introduces and formally specifies and verifies several standard topics in modal logic such as relationships between the standard modal axioms and properties of the accessibility relation, and attributes of the Barcan Formula and its converse in both constant and varying domains.

Contents

List of Figures

1 Introduction

The motivation for modal logics is to reason about various *modes* of truth: for example, what it means for something to be *possibly* true, or to *know* that something is true as opposed to merely *believing* it. The modal *qualifier* $\Box$ and its dual $\Diamond$ (defined as $\neg\Box\neg$) are used to indicate expressions that should be interpreted modally. All modal logics share the same basic structure but they employ different sets of axioms and make other adjustments according to the mode attributed to the qualifiers. For example, in an *Epistemic* modal logic, where $\Box$ is interpreted as knowledge, we will expect the formula $\Box P \supset P$ to hold: if I *know* that something is true, then it should be true (a traditional definition of knowledge is justified *true* belief). But we would not expect this formula to hold in a *Doxastic* logic, where $\Box$ is interpreted as belief. Instead, we might expect $\Box P \supset \Diamond P$ to hold: if I *believe* P is true, then I cannot also believe it to be false (reading $\Diamond P$ as $\neg\Box\neg P$). There is a collection of common formulas such as these that have standard names (the two above are called T and D, respectively) and that are used in various combinations to axiomatize different modal logics.

Standard modal logics are obtained by enriching either propositional logic or quantified (i.e,, first- or higher-order) logic to yield propositional and quantified modal logics, respectively. The semantics of modal logics are derived from those of classical logics by interpreting all expressions relative to a set of *possible worlds*. Thus, whereas a constant x has some fixed interpretation in a classical logic, in a modal logic its interpretation depends on the world w and this can be encoded or *embedded* in classical logic by *lifting* x to a function on worlds: $x(w)$ then denotes the interpretation of x in world w. The qualifier $\Box$ is interpreted as truth in *all* possible worlds, so $\Box P$ is $\forall w : P(w)$, while $\Diamond P$ applies to *some* possible worlds: $\exists w : P(w)$. There are complexities in the details because some worlds may not be *accessible* from others, and it is this that distinguishes the different modes from each other (and is closely related to the choice of axioms such as T and D).

In the next section, I will describe more of propositional modal logic and show how this can be embedded within PVS [8]. The embedding uses the resources of PVS in a way that allows modal formulas to be written in their standard syntax and used in combination with the other features of PVS (for example, its rich type system and comprehensive language for expressions and definitions). Proofs of modal formulas use the standard capabilities of the PVS prover and can be highly automated.

In the section following that, I describe quantified modal logic and its embedding in PVS. This embedding is only a slight extension of that for propositional modal logic, but the combination of modal logic and quantification has some intrinsic complexities that I describe and illustrate. In the final section, I discuss some topics in the pragmatics of constructing specifications in quantified modal logic, particularly the combination of modal and classical quantifiers.

2 Propositional Modal Logic in PVS

As mentioned in the introduction, the basic idea of the possible worlds interpretation for propositional modal logic is that expressions of propositional logic are *lifted* and interpreted relative to a possible world. Thus, a generic expression P in propositional logic must be transformed into its lifted form $l(P)$, which is a function that can be applied to a world w to deliver its value $l(P)(w)$ in that world. The lifting transformation is defined recursively over the constructs of propositional logic: that is, for a propositional connective such as $\wedge$, we define how $l(P \wedge Q)(w)$ is related to $l(P)(w)$ and $l(Q)(w)$, and similarly for other constructs. This is done below.

- Constants a or variables x (these are equivalent in propositional logic), are lifted by a *valuation* function V where $V(a)(w)$ and $V(x)(w)$ provide their values in world w.
- Negation is lifted by negating the lifted term: $l(\neg P)(w)$ is $\neg l(P)(w)$.
- Conjunction is lifted by conjoining its lifted conjuncts: $l(P \wedge Q)(w)$ is $l(P)(w) \wedge l(Q)(w)$. The other binary connectives are lifted in the same way.

We then specify the modal qualifiers as follows.

- $l(\Box P)$ is $\forall w : l(P)(w)$, where w is a fresh variable.
- $l(\Diamond P)$ is $\exists w : l(P)(w)$, where w is a fresh variable.
  (An alternative, but equivalent, interpretation is that $\Diamond P$ is $\neg\Box\neg P$)

Finally, a modal sentence P is valid if it is true in all possible worlds; that is, $\forall w : l(P)(w)$.

To illustrate these notions, I will use the following simple modal argument.[1] This uses an *Alethic* modal logic where the qualifiers are interpreted as necessity ($\Box$) and possibility ($\Diamond$). Interpretation of the modal qualifiers is related to an *accessibility relation* on worlds and corresponding axioms. For simplicity of exposition, I delay these topics to Section 2.3, but the elementary treatment (i.e., without an accessibility relation) used in the subsections before then is sound for Alethic logics (it is equivalent to the logic known as S5).

Notation: g is a constant, P is a metavariable.

Premise H1: $\Diamond g$ (i.e., g is possible)

Premise H2: $P \supset \Box P$ (i.e., that which is true is necessarily true).

Conclusion HC: g (i.e., g is true in the classical sense).

[1] This is actually a flawed version of Hartshorne's rendition of Anselm's *Proslogion* III argument for the existence of God [4, Section 4.1]. The flaw is use of a variable P in H2 where the actual argument uses g. Both forms of the premise are questionable but the one used here is worse; I employ it to illustrate aspects of propositional modal logic and its embedding. Those interested in the actual argument are referred to [10].

Each of these modal sentences is interpreted as the validity of its lifted form, so premise H1 becomes $\forall w : l(\Diamond g)$, which becomes $\forall w : \exists v : l(g)(v)$, which becomes $\forall w : \exists v : V(g)(v)$, the translation shown below. The outermost quantifier is superfluous in this case.

H1: $\forall w : \exists v : V(g)(v)$

H2: $\forall w : P(w) \supset (\forall v : P(v))$

HC: $\forall w : V(g)(w)$

H2 is similarly translated by recursively traversing its parse tree. We start with modal validity of $P \supset \Box P$, which becomes $\forall w : l(P \supset \Box P)(w)$; processing the $\supset$ connective, this becomes $\forall w : l(P)(w) \supset l(\Box P)(w)$, and then interpretation of the $\Box$ qualifier yields $\forall w : (l(P)(w) \supset \forall v : l(P)(v))$; lifted metavariables are themselves, so we end with the translation shown above. HC likewise becomes $\forall w : l(g)(w)$, which becomes $\forall w : V(g)(w)$, as shown above.

```
eg_direct: THEORY
BEGIN
  worlds: TYPE+
  pmlformulas: TYPE = [worlds -> bool]
  pvars: TYPE+

  v, w: VAR worlds
  x: VAR pvars

  val(x)(w): bool

  g: pvars
  P: VAR pmlformulas

  H1: AXIOM EXISTS w: val(g)(w)

  H2: AXIOM P(w) IMPLIES FORALL v: P(v)

  HC: THEOREM val(g)(w)

END eg_direct
```

Figure 1: Direct Shallow Embedding in PVS of Propositional Example

Figure 1 presents a direct transliteration into PVS of the possible worlds interpretation of the example argument that we constructed above. The type `pmlformulas` (for propositional modal logic formulas) is used for lifted formulas; its variables (such as `P`) function

as metavariables of the embedded logic. The type `pvars` is used for propositional variables (and constants), with `val` as their valuation function (cf. V in the mathematical rendition above); `g` is one of these propositional constants. We then state the lifted renditions of the premises and conclusion of the argument, exploiting the fact that PVS automatically applies universal closure to its formulas.

PVS proves the theorem HC automatically given the two premises.

PVS proof

```
(grind-with-lemmas :polarity? t :lemmas ("H1" "H2"))
```

An alternative way to state the theorem is to use an arbitrary world "`here`."

PVS fragment

```
here: worlds

HC_alt: THEOREM val(g)(here)
```

The same PVS proof as before will prove this alternative statement of the theorem.

2.1 Elementary Shallow Embedding of Propositional Modal Logic in PVS

The kind of transformation from one logic or language to another seen here is referred to as a *shallow embedding* [3]. The characteristic of a shallow embedding is that it is a syntactic transformation: the modal presentation of the example argument on page 2 is translated into the PVS specification shown in Figure 1. It would seem, therefore, that automating the transformation will require a syntax-to-syntax translator. Fortunately, the capabilities of PVS allow us to accomplish the translation quite effectively within PVS itself. This is feasible because the source language, propositional modal logic, is a *logic* and has much of its syntax in common with PVS; the techniques we are about to see would be less effective if the source were, say, a programming language.

The idea is to define the operators and connectives of the embedding of propositional modal logic directly in their lifted form. Thus, whereas we earlier defined the lifted form of conjunction $l(P \wedge Q)(w)$ to be $l(P)(w) \wedge l(Q)(w)$, here we will define a new modal conjunction operator $\&_m$ by $(P\&_m Q)(w) = P(w) \wedge Q(w)$. PVS allows function symbols and names to be *overloaded* (that is, the same symbol or name can be used for several different functions) and types are used to resolve the correct instance. Thus, in PVS we do not need a separate function name $\&_m$, we simply overload the existing & (whose built-in definition is a synonym for Boolean `AND`).[2]

This is done in the theory `shallow_pml` (for shallow propositional modal logic) shown in Figure 2. The first few blocks of declarations are the same as in `direct_hart`, then we define the lifted connectives ~, &, and => in the manner described above. Obviously, other

[2]Likewise ~ is already defined as a synonym for `NOT` and => as a synonym for `IMPLIES`; for ease of human parsing, we use symbols for the lifted operators and ASCII for the Boolean ones.

lifted propositional connectives can be added here in a similar way, or they can be defined in terms of those already defined. Notice that we are using symbols (e.g., &) as function names here; the PVS Language reference specifies the symbols that may be defined in this way [7, Figure 2.4].

```
elem_shallow_pml: THEORY
BEGIN

  worlds: TYPE+
  pmlformulas: TYPE = [worlds -> bool]
  pvars: TYPE+

  v, w: VAR worlds
  x, y: VAR pvars
  P, Q: VAR pmlformulas

  val(x)(w): bool

  ~(P)(w): bool = NOT P(w) ;
  &(P, Q)(w): bool = P(w) AND Q(w) ;
  =>(P, Q)(w): bool = P(w) IMPLIES Q(w) ;
  □(P)(w): bool = FORALL v: P(v) ;

  <>(P): pmlformulas = ~ □ ~ P ;

  |=(w, P): bool = P(w)
  valid(P): bool = FORALL w: w |= P

END elem_shallow_pml
```

Figure 2: Elementary Shallow Embedding of Propositional Modal Logic in PVS

Next, we define the □ qualifier of modal logic. The PVS Language reference suggests we could use `[]` here (as we use <> below), but recent versions of PVS preempt this syntax for declaration-level parameterization. However, these recent versions of PVS also allow use of Unicode, so we simply employ the appropriate Unicode character (□, hexadecimal code `25A1`) as the name of our function. Next, we define <> (we could have used ◇, Unicode `25C7` instead) as the dual modal qualifier. We could have done this in a similar way to □ (but using an existential quantifier), but for variety we will instead define it in terms of □. Notice that no parentheses are used here (i.e., we do not need ~(□(~(P)))). This is

because ~, □, and <> are known to be *unary* operators [7, Figure 7.1].[3] For symmetry with later constructions, we define `w |= P` to be truth of `P` in world `w`. Notice that `|=`, together with `&` and `=>`, are known to PVS as binary infix operators [7, Figure 7.1], although they must appear in prefix form when being defined. Finally, we define modal validity in the expected way.

Now that we have a shallow embedding of propositional modal logic in PVS, we can simply import it into a new theory where we specify the example argument in a fairly direct way. This is shown in Figure 3; the theorem is proved automatically in the same way as before. Notice that the modalities □ and <> are used here in a natural way because, as noted before, PVS treats these symbols as unary operators.

```
eg_elem_shallow1: THEORY
BEGIN IMPORTING elem_shallow_pml

  g: pvars
  P: var pmlformulas

  H1: AXIOM valid(<> val(g))

  H2: AXIOM valid(P => □ P)

  HC: THEOREM valid(val(g))

END eg_elem_shallow1
```

Figure 3: Partially Automated Shallow PVS Embedding of Propositional Example

All the formulas in Figure 3 explicitly employ the function `valid` to reduce validity of modal sentences to the classical validity employed in PVS. It would be nice to automate this and PVS provides a way to do it: we specify that `valid` is a `CONVERSION`. When a PVS expression fails to typecheck, PVS searches for a conversion function that will make it type-correct and applies it automatically (PVS provides commands that prettyprint the specification with conversions applied, and these are also shown expanded in proofs). Use of the conversion allows `H2` to be written in the standard modal syntax. If we additionally specify `val` as a conversion then `H1` also can be written in the standard syntax. However, the conclusion `HC` still requires verbose syntax: we would like to write just `g`. The reason we cannot do this is that it requires application of two conversions before `g` becomes type correct. We can define a function `validval` that applies both `valid` and `val` and declare that

[3] My actual recommendation is to use the definition `<>(P)(w): bool = EXISTS v: P(v)` because it is much easier to interpret the existential quantifier than the doubly negated universal when guiding interactive PVS proofs.

to be a conversion. Use of all three conversions allows us to simplify the PVS specification of the example argument to the form shown in Figure 4.

```
eg_elem_shallow2: THEORY
BEGIN IMPORTING elem_shallow_pml

  g: pvars
  P: var pmlformulas

  validval(x: pvars): bool = valid(val(x))
  CONVERSION valid, val, validval

  H1: AXIOM <> g

  H2: AXIOM P => □ P

  HC: THEOREM g

END eg_elem_shallow2
```

Figure 4: More Automated Shallow PVS Embedding of Propositional Example

It will generally be more convenient to specify these conversions in the theory `elem_shallow_pml` as they will then automatically be available wherever this is imported. The theorem `HC` is proved automatically in the same way as before—for, semantically, it is the same as the previous versions; the automation simply allows better syntax.

Notice that if PVS did not allow overloading of built-in and infix symbols, then the middle block of definitions in Figure 2 would have to be written using standard functional notation as follows.

PVS fragment

```
  mneg(P)(w): bool = NOT P(w) ;
  mand(P, Q)(w): bool = P(w) AND Q(w) ;
  mimp(P, Q)(w): bool = P(w) IMPLIES Q(w) ;
  mbox(P)(w): bool = FORALL v: P(v) ;
  mdia(P): pmlformulas = mneg(mbox(mneg(P)))
```

The premises of the example argument would then appear like this.

PVS fragment

```
  H1: AXIOM valid(mdia(val(g)))

  H2: AXIOM valid(mimp(P, mbox(P)))
```

The improvement in Figure 4 is obvious.

Now that we know of *shallow* embeddings, it will come as no surprise that there are *deep* embeddings also. I introduce these in the next subsection.

2.2 Elementary Deep Embedding of Propositional Modal Logic in PVS

Whereas shallow embeddings work on the surface syntax of the source language, deep embeddings are defined on its abstract syntax: we define an abstract datatype that models this syntax, then define functions that operate on it by recursion and case analysis [3]. The abstract syntax for propositional modal logic is specified by the PVS datatype in Figure 5.

```
modalformula[pvars: TYPE+]: DATATYPE
BEGIN
  pvar(arg: pvars): var?
  ~(arg: modalformula): not?
  &(arg1: modalformula, arg2: modalformula): and?
  =>(arg1: modalformula, arg2: modalformula): imp?
  □(arg: modalformula): box?
END modalformula
```

Figure 5: PVS Datatype for Abstract Syntax of Propositional Modal Logic

This abstract datatype defines the `modalformula` datatype, parameterized by the nonempty type `pvars` that specifies its propositional variables. We then import this into the theory `elem_deep_pml` shown in in Figure 7 where we define validity of `modalformula P` in world `w`, `w |= P`, by recursion on the structure of `P`. PVS ensures termination of recursive definitions by generating proof obligations and the `MEASURE` keyword instructs PVS to use the subterm ordering relation << in this proof obligation. The rest of the construction is the same as in `shallow_pml`.

We can then employ `elem_deep_pml` in the representation of the example argument shown in Figure 6. Notice that syntactically this identical to the representation using a shallow embedding that we saw in Figure 4.

The proof of the theorem is shown below. It installs the two premises, instantiates one of them, then applies the standard top-level proof strategy of PVS. This is not quite as automated as the proof using a shallow embedding but this is due to the rather weak quantifier instantiation heuristics in PVS rather than intrinsic difficulty.

PVS proof

```
(lemma "H1") (lemma "H2") (inst?) (grind :polarity? t)
```

For the purposes described here, there is little to choose between shallow and deep embeddings. For the remainder of this tutorial I will mostly use the shallow embedding, because it is somewhat simpler. A deep embedding would be preferable if we wanted to establish metalogical properties (i.e., properties *about* modal logic as opposed to properties stated *in* modal logic).

```
eg_elem_deep: THEORY
BEGIN IMPORTING elem_deep_pml

  g: pvars
  P: VAR modalformula

  H1: AXIOM <> g

  H2: AXIOM P => □ P

  HC: THEOREM g

END eg_elem_deep
```

Figure 6: Representation of the Propositional Example Using Deep Embedding in PVS

All our embeddings have so far implicitly assumed that all possible worlds are accessible from each other. This is an oversimplification[4] that we correct in the following section.

2.3 Shallow Embedding of Full Propositional Modal Logic in PVS

The modal qualifiers allow us to speak about truth in other possible worlds: when we say $\Diamond P$ we are saying that there are possible worlds where P is true. But suppose there is a structure on possible worlds and we cannot reach every possible world from every other one; then it might be that P is true only in worlds we cannot reach, so $\Diamond P$ would be false in this configuration of worlds. A full formalization of propositional modal logic uses an *accessibility relation* on possible worlds and adjusts the semantics of the modal qualifiers so they apply only to accessible worlds. In the shallow PVS embedding of Figure 2, we do this as follows, where `access` is a new declaration that introduces the accessibility relation and the definition of □ is modified to reference this. There is no need to modify the definition of <> if this is given in terms of □, but if it is defined directly, it takes the form shown below. We can make exactly analogous adjustments to the deep embedding.

PVS fragment

```
  access(w, v): bool

  □(P)(w): bool = FORALL v: access(w, v) IMPLIES P(v) ;
  <>(P)(w): bool = EXISTS v: access(w, v) AND P(v) ;
```

To complete our full shallow and deep PVS embeddings of propositional modal logic, we need to parameterize the theories. Currently the type `worlds`, its accessibility relation

[4] Actually, it does correspond to a legitimate modal logic—a variant on S5—that is described later.

```
elem_deep_pml: THEORY
BEGIN

  worlds, pvars: TYPE+
  IMPORTING modalformula[pvars]

  w, v: VAR worlds
  x, y: VAR pvars
  P, Q: VAR modalformula

  val(x)(w): bool

  |=(w, P): RECURSIVE bool =
     CASES P OF
      pvar(x): val(x)(w),
      ~(R): NOT (w |= R),
      &(R, S): (w |= R) AND (w |= S),
      =>(R, S): (w |= R) IMPLIES (w |= S),
      □(R): FORALL v: (v |= R)
    ENDCASES
  MEASURE P by <<

  <>(P): modalformula = ~ □ ~ P ;

  valid(P): bool = FORALL w: w |= P
  validval(x: pvars): bool = valid(pvar(x))
  CONVERSION valid, pvar, validval

END elem_deep_pml
```

Figure 7: Elementary Deep Embedding of Propositional Modal Logic in PVS

`access`, and the type `pvars` and its valuation function `val` are defined within the embedding theories. We need instead to specify some or all of these these as parameters, so that they can be defined by the theories that use the embedding. It is a matter of choice which of these types and constants are defined as parameters, but there are some dependencies. Certainly `pvars`, the type of the propositional variables, should almost certainly be a parameter. We will see a circumstance later where `val`, the valuation function needs to be a parameter; its type is `[pvars -> [worlds -> bool]]` so this forces both `pvars` and `worlds` to be parameters, in which case we may as well complete the parameterization by adding `access`.

This is shown in Figure 8, which is a fully parameterized version of the shallow embedding for full propositional modal logic. We name this modified PVS specification

```
full_shallow_pml [worlds: TYPE+, access: pred[[worlds,worlds]],
                  pvars: TYPE+, val: [pvars -> [worlds -> bool]]]: THEORY
BEGIN

  pmlformulas: TYPE = [worlds -> bool]
  v, w: VAR worlds
  x, y: VAR pvars
  P, Q: VAR pmlformulas

  ~(P)(w): bool = NOT P(w) ;
  &(P, Q)(w): bool = P(w) AND Q(w) ;
  =>(P, Q)(w): bool = P(w) IMPLIES Q(w) ;

   □(P)(w): bool = FORALL v: access(w, v) IMPLIES P(v) ;
  <>(P)(w): bool = EXISTS v: access(w, v) AND P(v) ;

  |=(w, P): bool = P(w)
  valid(P): bool = FORALL w: w |= P

  validval(x: pvars): bool = valid(val(x))
  CONVERSION valid, val, validval

END full_shallow_pml
```

Figure 8: Full Shallow Embedding of Propositional Modal Logic in PVS

`full_shallow_pml` and use it in the version of the example argument shown in Figure 9. The statement of the theorem is adjusted to record the fact that it requires the accessibility relation to be symmetric. The `full_shallow_pml` theory specifies `valid`, `val`, and `validval` as conversions.

As in the previous version, the proof requires some manual steps to guide the quantifier instantiation.

PVS proof

```
(grind-with-lemmas :lemmas ("H1" "H2"))
(inst -1 "val(g)")
(inst? -2)
(grind :polarity? t)
```

There is a correspondence between properties of the accessibility relation and certain modal formulas: for example, there is a modal formula that exactly corresponds to the accessibility relation being symmetric. We examine this topic in the next section.

```
eg_full_shallow: THEORY
BEGIN

  worlds, pvars: TYPE+
  access: pred[[worlds, worlds]]
  val(x:pvars)(w:worlds): bool

  IMPORTING full_shallow_pml[worlds, access, pvars, val]

  g: pvars
  P: var pmlformulas

  H1: AXIOM <> g

  H2: AXIOM P => □ P

  HC: THEOREM symmetric?(access) => g

END eg_full_shallow
```

Figure 9: The Propositional Example Using an Embedding of Full Propositional Modal Logic

2.4 Standard Axioms and Various Modal Logics

As we noted in the introduction, different modal logics apply different interpretations to the modal qualifiers and employ different sets of axioms to characterize their properties. There is a collection of standard axioms with single-character names that are used in various combinations to axiomatize different modal logics. The most widely used axioms are the following.

K: $\Box(P \supset Q) \supset (\Box P \supset \Box Q)$,

T: $\Box P \supset P$,

4: $\Box P \supset \Box\Box P$,

B: $P \supset \Box\Diamond P$,

D: $\Box P \supset \Diamond P$,

5: $\Diamond P \supset \Box\Diamond P$.

In addition to these axioms, there is a principle of *necessitation*, generally named N.

N: if P is valid, so is $\Box P$.

Necessitation is actually a metatheorem, true in all modal logics; it says that if P is just plain true (i.e., without reference to a possible world), then it is true in all possible worlds.

The different modal logics incorporate different sets of axioms, which are indicated by juxtaposing their names; thus a standard Deontic (obligation) logic uses **KD**, Doxastic (belief) logic uses **KD45**, while Epistemic (knowledge), and Alethic (necessity) logics use **KT45**, which is also known as **S5**.[5] Elementary logics of time use **KT4**, which is also known as **S4**, but the temporal logics in computer science such as LTL and CTL have rather more structure (they add a *next* operator). Notice that **K** is always present: it is actually a theorem, true in all modal logics, rather than an axiom.

It is a remarkable fact that each of the axioms above corresponds to a fairly natural property of the accessibility relation (apart from N and K, which are always true).

T: the accessibility relation is reflexive

4: the accessibility relation is transitive

B: the accessibility relation is symmetric

D: the accessibility relation is serial

5: the accessibility relation is Euclidean

If we denote the accessibility relation by R, the serial property is $\forall w : \exists v : R(w, v)$ while Euclidean is $\forall u, v, w : R(u, v) \wedge R(u, w) \supset R(v, w)$. A relation that is both symmetric and Euclidean is also transitive, and a relation that is both reflexive and Euclidean is also symmetric and hence transitive and, therefore, an equivalence relation. Thus, the accessibility relations of modal logics that include axioms T and 5 are equivalence relations.

Another concept that uses modal constructs is *strict* implication, usually written $\dashv\!\exists$, in contrast to the *material* implication $\supset$ of classical logic. We say that P strictly implies Q if it is not possible for P to be true and Q false: that is, in an Alethic modal logic

$$P \dashv\!\exists\ Q \stackrel{\text{def}}{=} \neg\Diamond(P \wedge \neg Q).$$

It is a theorem of Alethic logic that strict implication is the same as necessary material implication:

$$P \dashv\!\exists\ Q = \Box(P \supset Q).$$[6]

[5] S5 differs from the modal logic introduced in Section 2.1 (where the accessibility relation was absent) in that its worlds can be structured as several isolated cliques, whereas in the logic of the earlier section all worlds implicitly belong to a single clique. However, the two logics have the same valid sentences and can be considered equivalent.

[6] This equality is a theorem of all modal logics (i.e., it requires no axioms) but it carries the intended interpretation only in Alethic logics.

We will now add these notions to the PVS embeddings of propositional modal logic and prove the various theorems. We use the shallow embedding for illustration, but the deep embedding is handled similarly.

First, we define strict implication (as the infix symbol `|>`) and the standard modal axioms. These are shown in Figure 10, preceded by a theory that defines serial and Euclidean relations and their properties, as these are lacking from the `relations` theory built in to the PVS Prelude. All the lemmas in these theories are proved automatically, apart from the very first, `sym_Euc`, which requires a manual quantifier instantiation step.

Some remarks on the modal axioms seem appropriate at this point. Those that are used in standard theories (i.e., TD45) seem appropriate to their intended interpretation; B is more controversial, however. It looks innocuous, but it is equivalent to $\Diamond\Box P \supset P$, which seems less so. Premises similar to H2 are generally justified by virtue of their similarity to N, the principle of necessitation, which is true in all modal logics. That similarity is superficial, however. If we invite PVS to expand N in its shallow embedding from the theory `modal_axioms` of Figure 10, it delivers the following

N in PVS

```
(FORALL u: P!1(u))
       IMPLIES (FORALL w: FORALL v: access(w, v) IMPLIES P!1(v))
```

whereas doing the same for H2 in `eg_full_shallow` of Figure 9 delivers the following.

H2 in PVS

```
(FORALL w: P!1(w)
       IMPLIES (FORALL v: access(w, v) IMPLIES P!1(v)))
```

The crucial difference in quantification (and parenthesization) is now quite evident. N says that if P is true in every world, then it must be true in every world that is accessible from a given world. This is obviously true (and provable). Whereas H2 says that if P is true in a given world, it must also be true in every world that is accessible from that world. This is a strong claim and one that seems difficult to justify in general.

This highlights another item worth noting, namely, that the deduction theorem is not valid in modal logic. That is to say, the following (which would allow `H2` to be derived from `N`) is not provable (its converse is, however).

PVS fragment

```
% false
  nondeduction: CLAIM (valid(P) IMPLIES valid(Q)) IMPLIES valid(P => Q)
```

Related to this is the observation that it is often necessary to be careful about which parts of a sentence are to be interpreted modally, and which are conventional propositional logic. Specifically, a correct statement of modal *modus tollens* is expressed in PVS as follows (i.e., two modal sentences connected propositionally),

PVS fragment

```
  tollens: LEMMA (P => Q) IMPLIES (□~Q => □~P)
```

```
more_relations [T: TYPE]: THEORY
BEGIN

  IMPORTING relations[T]
  R: VAR PRED[[T, T]]
  x, y, z: VAR T

  serial?(R):    bool = FORALL x: EXISTS y: R(x, y)
  Euclidean?(R): bool = FORALL x, y, z: R(x, y) & R(x, z) => R(y, z)

  sym_Euc:  LEMMA symmetric?(R) AND Euclidean?(R) IMPLIES transitive?(R)
  ref_Euc1: LEMMA reflexive?(R) AND Euclidean?(R) IMPLIES symmetric?(R)
  ref_Euc2: LEMMA reflexive?(R) AND Euclidean?(R) IMPLIES equivalence?(R)

  equiv_is_Euclidean: JUDGEMENT (equivalence?) SUBTYPE_OF (Euclidean?)

END more_relations

modal_axioms: THEORY
BEGIN

  worlds, pvars: TYPE+
  val: [pvars -> [worlds -> bool]]
  access: pred[[worlds,worlds]]
  IMPORTING full_deep_pml [worlds, access, pvars, val]
  IMPORTING more_relations[worlds]

  P, Q: VAR pmlformulas

  % strict implication
  |>(P, Q): pmlformulas = ~<>(P & ~Q)

  strict_material: LEMMA P |> Q = □(P => Q)

  K: LEMMA □(P => Q) IMPLIES (□P => □Q)
  N: LEMMA valid(P) IMPLIES valid(□P)

  T: AXIOM □ P => P
  four: AXIOM □ P => □ □ P
  B: AXIOM P => □ <> P
  D: AXIOM □ P => <> P
  five: AXIOM <> P => □ <> P

END modal_axioms
```

Figure 10: Strict Implication and the Standard Modal Axioms in PVS

whereas the following (i.e., a single modal sentence) is invalid.

PVS fragment

```
tollens_bad: CLAIM (P => Q) => (□~Q => □~P)
```

Next, we prove that the standard modal axioms are each equivalent to a property of the accessibility relation. We do this in two stages. First we prove that properties of the accessibility relation entail the corresponding modal axiom. This is done in the theory `modal_props` shown in Figure 11. The formal specification here is a rather unsatisfactory. What we would like to say is something like the following

Formerly not PVS

```
T_refl: LEMMA reflexive?(access) IMPLIES T
```

where we reference the name of one formula (here, `T`) within another formula. Unfortunately, PVS does not support this,[7] so we have to repeat the actual formula named by `T`. All the lemmas are proved automatically, except the last two, which require manual quantifier instantiation.

```
modal_props: THEORY
BEGIN

  IMPORTING modal_axioms
  IMPORTING more_relations[worlds]

  P: VAR pmlformulas

  T_refl: LEMMA reflexive?(access) IMPLIES □ P => P

  four_trans: LEMMA transitive?(access) IMPLIES □ P => □ □ P

  B_sym: LEMMA symmetric?(access) IMPLIES P => □ <> P

  D_serial: LEMMA serial?(access) IMPLIES □ P => <> P

  five_Euc: LEMMA Euclidean?(access) IMPLIES <> P => □ <> P

END modal_props
```

Figure 11: Accessibility Relation Properties and the Standard Modal Axioms in PVS

In the final stage, we prove that each of the modal axioms entails a property of the accessibility relation (i.e., the reverse direction of the previous stage). By comparison with the previous theory, it might seem that we would state these lemmas in a similar manner, as shown below.

[7] Recent versions of PVS do support this, so `T_refl` is now valid PVS.

PVS fragment

```
refl_T: LEMMA □ P => P IMPLIES reflexive?(access)
```

However there is a complication. To introduce this, note that the informal way to prove the result is via its contrapositive: we suppose the accessibility relation is not reflexive, so there is some world `w` such that `NOT access(w, w)`. Now define the valuation function `val` such that `val(p)(v) = NOT v=w` for some `pvar p`. Then `w |= pvar(p)` is false but `w |= □ pvar(p)` is true (because the valuation of `p` is true everywhere except at `w` and `w` is not accessible from `w`). Thus, we contradict the antecedent and by *reducto* conclude that the accessibility relation must be reflexive. This is a sound proof because `□ P => P` must be true in all models.

```
more_modal_props: THEORY
BEGIN

  worlds, pvars: TYPE+
  val: VAR [pvars -> [worlds -> bool]]
  access: pred[[worlds,worlds]]
  IMPORTING full_deep_pml
  IMPORTING more_relations[worlds]

  P: VAR modalformula[pvars]

  refl_T: LEMMA (FORALL P, val:
     full_deep_pml[worlds,access,pvars,val].valid(□P => P))
    IMPLIES reflexive?(access)

END more_modal_props
```

Figure 12: Accessibility Relation Properties and the Standard Modal Axioms (other direction) in PVS

Now the issue in constructing this proof in PVS is that we need to be able to define a suitable valuation function during the proof: thus the function needs to be a variable. In all our specifications so far we have defined the parameters to the embedding as uninterpreted constants. Here, we need to keep the valuation function `val` as a free variable—hence, the quantification in the specification of the formula `refl_T` shown in Figure 12. Unlike earlier specifications, the conversion `valid` cannot be applied automatically as PVS cannot tell what theory instance is required, so we have to specify it in its qualified form, where we state the full theory theory instance. It is likewise difficult to specify the type of `P`. We cannot supply `val` as a parameter to its theory instance as this is a variable, so we need to reparameterize the embedding so that modal formulas are specified separately from their interpretation. Rather than do this for `full_shallow_pml`, we use `full_deep_pml` as it is already structured and parameterized in this way.

The proof of `refl_T` is shown below. The `(lemma "pvars_nonempty")` is used to access the name (`x!2`) of some member of `pvars` (since this type is specified to be nonempty). The `inst` command then constructs the valuation function described above, after which PVS can complete the proof automatically.

PVS proof

```
(ground)
(expand "reflexive?")
(skosimp)
(lemma "pvars_nonempty")
(skosimp)
(inst - "pvar(x!2)"
  "LAMBDA (x:pvars): LAMBDA (w:worlds): NOT (x=x!2 AND w=x!1)")
(grind)
```

The theorems and proofs for the other modal axioms are constructed similarly, but are sufficiently ~~tedious~~ challenging that they are left as exercises for the reader. Another exercise is to rework the version of the example argument in Figure 9 so that it cites modal Axiom B instead of symmetry of the accessibility relation.

2.5 LaTeX-Printing

PVS is able to typeset specifications in LaTeX and thereby reproduce standard mathematical notation. We illustrate this on the following example specification that uses strict implication.

Example Specification

```
|>(P, Q): pmlformulas =  ~<>(P & ~Q)

H1: AXIOM g |> □g

H_triv: LEMMA symmetric?(access) IMPLIES (P |> □P) IMPLIES (<>P => P)

H2: AXIOM <>g

HC: THEOREM symmetric?(access) => g
```

A LaTeX-printed version of this example is shown below.

$P \dashv\!\!\exists\, Q$: pmlformulas $\stackrel{\text{def}}{=} \neg\Diamond(P \wedge \neg Q)$

H1: **AXIOM** $g \dashv\!\!\exists\, \Box g$

H1_triv: **LEMMA** symmetric?(access) $\supset$ $(P \dashv\!\!\exists\, \Box P) \supset (\Diamond P \supset P)$

H2: **AXIOM** $\Diamond g$

HC: **THEOREM** symmetric?(access) $\supset$ g

The way in which PVS ASCII text is rendered in LaTeX is controlled by a "substitutions" file `pvs-tex.sub`. A comprehensive substitution file is provided with the PVS distribution but it can be augmented by the user. The rendition above was generated using the following substitutions file as augmentation. The first column in this file identifies the ASCII source to be substituted, the second identifies the kind of PVS object it is (see the PVS documentation), the third gives the size of the substitution in *em*s, and the final column gives the desired LaTeX substitution.

PVS LaTeX substitution file

```
|>              2       3       {#1 \strictif #2}
|>             id       1       \strictif
H1             id       2       \pvsid{H1}
H1_triv        id       3       \pvsid{H1\_triv}
H2             id       2       \pvsid{H2}
IMPLIES        id       3       \supset
=             key       2       \defn
~              id       1       \sim
~              id       1       \neg
=>             id       1       \supset
&              id       1       \wedge
```

It might be considered that these substitutions are too aggressive because they fail to distinguish those connectives that are to be interpreted modally from those that are propositional. If we remove the last three lines from the substitution file above (where the third-last line was overriding the fourth-last), we restore this distinction and generate the following rendition.

$P \dashv\!\exists\, Q$: pmlformulas $\stackrel{\text{def}}{=} \sim\Diamond(P \ \& \ \sim Q)$

H1: **AXIOM** $g \dashv\!\exists\, \Box g$

H1_triv: **LEMMA** symmetric?(access) $\supset$ $(P \dashv\!\exists\, \Box P) \supset (\Diamond P \Rightarrow P)$

H2: **AXIOM** $\Diamond g$

HC: **THEOREM** symmetric?(access) $\Rightarrow$ g

We have now completed our treatment of propositional modal logic in PVS and proceed to tackle the quantified case.

3 Quantified Modal Logic in PVS

Quantified modal logics add quantification to the propositional case. There can be interactions between quantifiers and the modal qualifiers so this needs to be done with care. For example, a standard step in modal formulations of of Anselm's *Proslogion* II argument for the existence of God [4] (the "Ontological Argument," not to be confused with his Proslogion III argument which we used for illustration in the previous section) is to consider "some thing x than which there is nothing greater." This might be formulated as $\neg\exists y : \Diamond(y > x)$, which can be read as "there is no y that is greater than x in any (accessible) possible world." A plausible alternative is $\neg\Diamond\exists y : (y > x)$, "in no (accessible) possible world is there a y greater than x" and we might wonder if this is equivalent to the previous formula. It turns out that sometimes it is and and sometimes it is not, thereby highlighting the delicacy of combining quantified and modal reasoning.

We saw that the full semantics for propositional modal logic must recognize that possible worlds may not all be accessible from each other, and this is formalized in the accessibility relation. In a similar manner, a precise semantics for quantified modal logic must recognize that the domains of quantification may not be the same in all possible worlds. In a *constant domains* semantics they are the same, whereas in a *varying domains* semantics they may differ. Important special cases are *nondecreasing* and *nonincreasing* varying domains where the domains behave as their names suggest across the accessibility relation (obviously, constant domains are a special case of both of these). The two formulas considered above are equivalent if constant domains are assumed; with varying domains, the first implies the second under nonincreasing domains and vice-versa under nondecreasing domains.

Just as properties of the accessibility relation correspond to standard modal formulas (e.g., symmetry corresponds to modal Axiom B), so properties of domains correspond to standard formulas. These are the **Barcan Formula**

$$\forall x : \Box\phi(x) \supset \Box\forall x : \phi(x)$$

or, equivalently, its contrapositive

$$\Diamond\exists x : \phi(x) \supset \exists x : \Diamond\phi(x)$$

and the **Converse Barcan Formula**

$$\Box\forall x : \phi(x) \supset \forall x : \Box\phi(x)$$

and its contrapositive equivalent

$$\exists x : \Diamond\phi(x) \supset \Diamond\exists x : \phi(x).$$

The Barcan formula characterizes nonincreasing domains, while the Converse Barcan formula characterizes nondecreasing domains; the two are equivalent if the accessibility relation is symmetric (i.e., if modal Axiom B holds).

The Barcan formula and nonincreasing domains are generally regarded with suspicion: they imply that anything that exists in a possible world also exists in the actual world, a position known as *actualism*. Thus, if it is possible that a cow jumped over the moon, then there is a specific cow in the actual world that possibly jumped over the moon. The converse Barcan formula is less controversial.

3.1 Constant Domains

To represent these topics in PVS, we begin with constant domain semantics. We take the full shallow embedding of Figure 8, rename `pmlformulas` to `qmlformulas`, and add the following declarations.

PVS fragment

```
QT: TYPE
qmlpreds: TYPE = [QT -> qmlformulas]
PP: VAR qmlpreds

CFORALL(PP)(w): bool = FORALL (x:QT): PP(x)(w)
CEXISTS(PP)(w): bool = EXISTS (x:QT): PP(x)(w)
```

Here, the type `QT` is the "domain" over which quantification occurs. In a finished formalization it will be best to specify this as a parameter to the embedding theory. In classical logic, quantification applies to predicates over the quantified type; in quantified modal logic, everything is lifted to operate over possible worlds, so quantification applies to `qmlpreds`, which are functions from the quantified type to predicates on worlds (i.e., `qmlformulas`).

We then specify the constant domain universal quantifier `CFORALL` as a function that takes a lifted predicate of type `qmlpreds` and delivers a lifted Boolean of type `qmlformulas`; the definition of this function is quite natural, it applies classical universal quantification to all values of the lifted predicate in the world concerned. The corresponding existential quantifier is specified similarly as `CEXISTS`.

We can now attempt to employ these definitions by stating and proving some properties of the Barcan formula. In a constant domain, both the Barcan formula and its converse are true, so we can prove that the two sides of the formula are equal. What we would like to write is the following.

Not yet PVS

```
Barcan: LEMMA CFORALL(□ PP) = □ CFORALL(PP)
```

Unfortunately, this is incorrect. The modal qualifier □ applies to `qmlformulas` and on the left we have given it `PP` of type `qmlpreds`; what would work here is `□ PP(s)` for some `s`. However, `CFORALL` applies to `qmlpreds` and `□ PP(s)` is of type `qmlformulas`; what we need here is a function that takes an `s` and returns `□ PP(s)`. Such a function is `LAMBDA s: □ PP(s)`, so a correct statement of the Barcan formula is the following (the right side of the equality was already type-correct).

PVS fragment

```
Barcan1: LEMMA
  CFORALL (LAMBDA (s:QT): □ PP(s)) = □ CFORALL(PP)
```

The formula is proved by the commands shown below; `apply-extensionality` proves two functions are equal by showing that their values are equal when applied to each element of their domain.

PVS proof

```
(skosimp) (apply-extensionality :hide? t) (grind :polarity? t)
```

Although `Barcan1` is correct, it is a little difficult to read and rather more difficult to write. Fortunately, PVS has capabilities that considerably ease these difficulties. First is the `K_conversion`; this is defined in the PVS prelude as the K combinator of Combinatory Logic (indeed, it might be less confusing if it were named `K_combinator`). When used as a conversion it automatically supplies the `LAMBDA` construction of `Barcan1`. Thus the following is a valid rendering of our original formulation.

PVS fragment

```
CONVERSION K_conversion

Barcan: LEMMA CFORALL(□ PP) = □ CFORALL(PP)
```

The second PVS capability that is useful here is its ability to use higher-order predicates as "binders." The binder corresponding to a higher-order predicate is its name with `!` appended, so the binder corresponding to `CFORALL` is `CFORALL!`. Thus another way to render the Barcan formula is the following.

PVS fragment

```
Barcan2: LEMMA
  (CFORALL! (s:QT): □ PP(s)) = □ CFORALL! (s:QT): PP(s)
```

The right hand side could also be written as just □ CFORALL(PP), as it was previously.

These alternative forms are purely syntactic variations and are interpreted the same as Barcan1 internally.

Now, it might seem that instead of saying the two sides of the Barcan formula are equal in constant domains, we could say each implies the other, using the IFF connective as follows.

PVS fragment

```
Barcan3: LEMMA CFORALL(□ PP) IFF □ CFORALL(PP)
```

However, if we ask PVS to display the expanded forms (i.e., with conversions applied) of Barcan and Barcan3, using the command M-x ppe we see they are different.

PVS PPE

```
Barcan: LEMMA CFORALL(LAMBDA(s: QT): □PP(s)) = □CFORALL(PP)

Barcan3: LEMMA
  valid(CFORALL(LAMBDA (s: QT): □PP(s))) IFF valid(□CFORALL(PP))
```

The reason is that all types have an equality operator, so Barcan (once conversions are applied) is type-correct as it stands. The connective IFF, however, requires arguments of Boolean type, so PVS searches for a conversion that will take the qmlformulas we have provided and deliver Booleans: it finds that valid fits the bill.

We can repeat this experiment using implication as in the following two examples.

PVS fragment

```
Barcan4: LEMMA CFORALL(□ PP) IMPLIES □ CFORALL(PP)

Barcan5: LEMMA CFORALL(□ PP) => □ CFORALL(PP)
```

The first uses the keyword IMPLIES and the second the operator =>. These are synonyms when given Boolean arguments, but we have overloaded => to take qmlformulas also. Thus, when we display their expanded forms, we see they are different,

PVS PPE

```
Barcan4: LEMMA
  valid(CFORALL(LAMBDA (s: QT): □PP(s))) IMPLIES valid(□CFORALL(PP))

Barcan5: LEMMA valid(CFORALL(LAMBDA (s: QT): □PP(s)) => □CFORALL(PP))
```

These different formulas are all true (and provable) in constant domains but have slightly different meanings. This raises the question: which is the correct interpretation of the Barcan formula? Most texts on the topic are quasi-formal and do not clearly resolve (or recognize) the different possible interpretations. My belief is that Barcan5 is the correct reading. It is worth repeating in this context our earlier observation (page 16) that the deduction theorem (which would relate Barcan4 and Barcan5) is not valid in modal logic.[8]

[8] Barcan4 and Barcan5 are equivalent in constant domains, but not in varying domains.

3.2 Varying Domains

So much for constant domains; the more general treatment of quantified modal logic uses varying domains. We can formalize this in PVS by introducing a higher order predicate `vind(w)` (standing for "values in domain") that defines the set of members of the type `QT` that are defined in world `w`. This predicate is best specified as a parameter to the theory. We then require that the quantifiers are restricted to just the defined values. The obvious way to do this is the following.

PVS fragment

```
vind(w)(a): bool

VFORALL(PP)(w): bool = FORALL (x:QT): vind(w)(x) IMPLIES PP(x)(w)
VEXISTS(PP)(w): bool = EXISTS (x:QT): vind(w)(x) AND PP(x)(w)
```

However, PVS has predicate subtypes, so the following is a better way to express this.

PVS fragment

```
VFORALL(PP)(w): bool = FORALL (x:(vind(w))): PP(x)(w)
VEXISTS(PP)(w): bool = EXISTS (x:(vind(w))): PP(x)(w)
```

Here, `vind(w)` is a predicate and a predicate in parentheses denotes the corresponding subtype; hence, `x:(vind(w))` indicates that variable `x` ranges over the subtype of values satisfying `vind(w)`.

As we know, there are relationships between the Barcan formulas and the way domains change across the accessibility relation. We define these as follows.

PVS fragment

```
fixed: AXIOM vind(w)(a)

nondecreasing: AXIOM (EXISTS w: vind(w)(a) AND access(w,v)) => vind(v)(a)

nonincreasing: AXIOM (EXISTS w: vind(w)(a) AND access(v,w)) => vind(v)(a)
```

The axiom `fixed` asserts that these varying domains are actually constant: every value is defined in every world. The axiom `nondecreasing` says that new values may be added as we move along the accessibility relation, but none are lost; nonincreasing says the opposite.

Incidentally, these two formulas would mean the same if written without explicit quantification, as follows.

PVS fragment

```
nondecreasing_alt: AXIOM vind(w)(a) AND access(w,v) => vind(v)(a)

nonincreasing_alt: AXIOM vind(w)(a) AND access(v,w) => vind(v)(a)
```

Most readers will know this, but as this is a tutorial it is worth spelling it out. PVS closes formulas by universally quantifying all free variables at the outermost level. Here, we

have implications where variables a and v are referenced on both sides, but w is referenced only on the left. If we wish to explicitly quantify w in its narrowest scope, we must use an existential, not a universal, because there is an implicit negation in the left side of an implication. Readers who are unfamiliar with this may wish to examine these formulas (and prove their equivalence) in the PVS prover.

The Barcan formula is equivalent to nonincreasing domains and the converse Barcan formula to nondecreasing. In one direction, we can simply state an instance of the Barcan formula and prove it by citing the relevant axiom on domains.

PVS fragment

```
% requires fixed
  vBarcan_eq: LEMMA VFORALL(□ PP) = □ VFORALL(PP)

% requires nonincreasing
  vBarcan: LEMMA VFORALL (□ PP) => □ VFORALL (PP)

% requires nondecreasing
  vCBarcan: LEMMA  □ VFORALL (PP) => VFORALL (□ PP)
```

These are all easily proved by citing the axiom concerned and then using grind as usual.

More interesting is to prove the relationships in the other direction. We do this below. Notice that as the Barcan formula is now part of a larger formula, we must be explicit about the scoping of the quantification of PP (previously we could just allow PVS to close the formula). As always, we must state the relevant formula (here, nonincreasing) rather than just use its name.

PVS fragment

```
  vBarcanx: LEMMA
    (FORALL PP: (VFORALL (□ PP) => □ VFORALL (PP))) IMPLIES
      (FORALL v,a: (EXISTS w: vind(w)(a) AND access(v,w)) => vind(v)(a))
```

A proof for this is shown below. First, we use a variant of grind that performs no quantifier instantiations. Then we supply a carefully crafted instantiation for PP that will lead to a contradiction (we have nested LAMBDAs because the predicate is higher-order), and then instantiate the other variables in a way that makes the contradiction manifest.

PVS proof

```
(grind :if-match nil)
(inst - "LAMBDA (z:QT): LAMBDA (w:worlds): NOT(z=a!1 AND w=w!1)")
(inst -1 "v!1")
(ground)
(inst -1 "w!1")
(grind)
```

For the Converse Barcan formula, we state the result as follows (note that we have changed the quantification for w, v and a in illustration of the point made earlier about quantification on the left side of implications).

PVS fragment

```
vCBarcanx: LEMMA
  (FORALL PP: □ VFORALL (PP) => VFORALL (□ PP)) IMPLIES
    (FORALL w, v, a: vind(w)(a) AND access(w,v) => vind(v)(a))
```

And the proof is shown below.

PVS proof

```
(grind :if-match nil)
(inst - "LAMBDA (z:QT): LAMBDA (w:worlds): NOT(z=a!1 AND w=v!1)")
(inst -1 "w!1")
(ground)
(("1" (inst - "a!1") (grind)) ("2" (grind)))
```

Given these results, we can prove that the Barcan and Converse Barcan are either both valid or both false when the accessibility relation is symmetric.

PVS fragment

```
bothB: LEMMA symmetric?(access) IMPLIES
  ((FORALL PP: VFORALL (□ PP) => □ VFORALL (PP)) IFF
     (FORALL PP: □ VFORALL (PP) => VFORALL (□ PP)))
```

We leave proof of this as an exercise; the basic approach is to first prove a similar result about the `nonincreasing` and `nondecreasing` formulas (PVS can prove this automatically), then use `VBarcanx` and `VCBarcanx` to extend this to the Barcan formulas (which requires about a dozen PVS proof steps).

Our treatment of modal logic in PVS is now adequately complete. We have an embedding of quantified modal logic and have seen stated and proved some of the standard results concerning properties of the accessibility relation and of constant and varying domains. We have seen how the capabilities of PVS allow modal formulas to be presented in a fairly standard and attractive way. To finish, we need to discuss some of the pragmatics of using these embeddings, including the notions of *rigid* versus *flexible* functions and predicates, and the use of quantification in mixed classical and modal contexts.

3.3 Pragmatics of Quantified Modal Logic in PVS

Eder and Ramharter [4, page 2,819] quote Bertrand Russell as follows:

> I have heard a touchy owner of a yacht to whom a guest, on first seeing it, remarked: 'I thought your yacht was larger than it is'; and the owner replied, 'No, my yacht is not larger than it is'.

The issue here is that of comparing the value of an attribute of an object in one world with its value in another possible world. Here, the guest is comparing the size of the yacht in the actual world with its size in the world of his imagination, while the owner is rooted in the actual world. This kind of comparison across worlds is a topic that arises quite often and there is a method for dealing with it that I will describe in this section, along with

some other topics that arise in the construction of more complex specifications involving quantified modal logic.

At the beginning of this section, we saw a couple of quantified modal formulas that purported to capture the idea of "some thing x than which there is nothing greater." One of them was $\neg\exists y : \Diamond(y > x)$, which can be read as "there is no y that is greater than x in any (accessible) possible world." The problem with this formulation is that y may be greater than x in some worlds, but its greatness in those worlds is exceeded by the greatness of x in the actual world. So what we really want to compare is the greatness of x in this, the *actual world*, against that of potential rivals in other possible worlds.

Eder and Ramharter propose the following definition [4, Section 4.2] for the predicate G that recognizes maximally great things under this new interpretation. Here $\mathfrak{g}(x)$ is the "greatness" of x and $\succ$ is an ordering (actually an uninterpreted predicate) on greatness.

Def M-God 3: $Gx :\leftrightarrow \exists z\,(z = \mathfrak{g}(x) \wedge \neg\Diamond\exists y\,(\mathfrak{g}(y) \succ z))$

The quantified variable z is used to capture the greatness of x in *this* world, so that it can be compared to that of some y in another possible world.

Before we look at the combination of modal and quantified constructions here, it is worth taking note of the functions and predicates involved. The greatness of a thing is different in different worlds. It is what is called a *flexible* function and its lifted form will be $\mathfrak{g}(x)(w)$, giving the greatness of x in world w. But $\succ$ is a fixed or *rigid* predicate: it does not depend on the world (and so its lifted form is just itself).[9] consider the quantification appearing in this definition. The first quantifier is over greatness, whereas the second is over things. Our PVS embedding of quantified modal logic was defined in a theory which took a domain of quantification as a parameter. Here we have two domains, so it looks as if we may need to extend the embedding. or find a way to combine two instances of the embedding theory. The latter will entail the presence of two different interpretations for `qmlformulas` and, depending how the parameterization is done, several other types, too. It is feasible to resolve these issues by supplying almost all types used in the `full_shallow_qml` theory as parameters, but it seems appropriate to first look for alternative solutions.

The whole purpose of the quantification over z is to provide a rigid value that can be compared to flexible ones; that is, to provide a nonmodal (i.e., classical) context for evaluation of modal expressions. So what we really want to write as the body of `MGod3`, our PVS rendition of M-God 3, is something like the following, where `z` is a classical quantification outside the modal expression. We use varying domains (i.e., `VEXISTS!`) for generality.

[9] Varying domains allow the possibility that a constant c in the domain of quantification does or does not exist in different worlds; a possible further complication is that it may exist but denote different objects in different worlds. Thus $\mathfrak{g}(c)$ could change from one world to another due to either or both the flexibility of $\mathfrak{g}$ or of c. Fitting and Mendelsohn give details [5]. Flexible constants add complexity to the embedding but seem to have little expressive value and we omit them (if c denotes a in some worlds and b in others, we can replace it by c_a and c_b; the former always denotes a and exists in worlds where c exists and denotes a, and mutatis mutandis for c_b).

Flawed PVS

```
MGod3(x): qmlformulas =
  EXISTS z: (z=g(x) & ~ <> VEXISTS! y: (g(y) > z))
```

There are two problems here: one is that the type of the expression quantified by z is `qmlformulas` whereas classical quantification requires a Boolean expression; the other is that `MGod3(x)` is declared to be of type `qmlformulas`, whereas the quantified expression is delivering a Boolean. We solve both problems by recognizing that `MGod3(x)(w)` (i.e., the evaluation of `MGod3(x)` in world `w`) is a Boolean and so is the modal expression quantified by `z` when applied to the world `w`. Hence we arrive at the following formalization.

PVS fragment

```
modal_eandr: THEORY
BEGIN
  things: TYPE+

  x, y: VAR things
  IMPORTING full_shallow_qml[things]
  w, v: VAR worlds

  greatness: TYPE+
  a, b, z: VAR greatness

  g(x)(w): greatness

  >(a, b): bool

  MGod3(x)(w): bool =
       EXISTS z: (z = g(x) & ~ <> VEXISTS! y: (g(y) > z))(w)
```

The theory defines `greatness` as an uninterpreted type; `g(x)(w)` is a flexible function that gives the `greatness` of `x` in world `w`, and > is the rigid ordering over `greatness` (although, since it is uninterpreted, nothing says it is really an ordering relation). Then we define `MGod3` in the manner described above.

But now notice that the subexpressions `z = g(x)` and `g(y) > z` in `MGod3` are not type-correct: `z` is of type `greatness` but `g(x)` and `g(y)` are functions from `worlds` to `greatness`. The subexpressions become correct when we lift them to functions on worlds. The `K_conversion` of PVS does this automatically, so the following is the prettyprint-expanded form of the definition above.

PVS PPE

```
MGod3(x)(w): bool =
    EXISTS z:
      ((LAMBDA (x1: worlds[U_beings]): z = g(x)(x1)) &
        ~ <>VEXISTS! y: (LAMBDA (s: worlds[U_beings]): g(y)(s) > z))(w)
```

Thus, although our PVS specification for `MGod3` is syntactically quite similar to Eder and Ramharter's definition M-God 3, we see that quite a lot of machinery, and thought, are required to create this similarity.

If we no longer require similarity to Eder and Ramharter's definition, then some simpler alternatives become available, such as the following.

PVS alternative

```
MGod3_alt(x)(w): bool =
    (~ <> VEXISTS! y: LAMBDA (s: worlds): (g(y)(s) > g(x)(w)))(w)
```

Here, we exploit the fact that we have the world `w` available and can therefore name the greatness of `x` in the current world directly as `g(x)(w)`, thereby obviating the need for the quantification over `z`. On the other hand, we have to lift the whole expression to a function on worlds using an explicit `LAMBDA`.

This version of Anselm's Ontological Argument uses three premises that Eder and Ramharter formulate as follows.

ExUnd: $\exists x Gx$,

PossEx: $\forall x \Diamond E!x$, and

Greater 5: $\forall x \forall y(\neg E!x \wedge \Diamond E!y \rightarrow \exists z\,(z = \mathfrak{g}(x) \wedge \Diamond(\mathfrak{g}(y) \succ z)))$,

where $E!x$ is a flexible predicate indicating that x "exists in reality." The conclusion is the following.

ERC: $\exists x(Gx \wedge E!x)$.

The first two premises and the conclusion are transcribed quite directly into PVS as shown below.

PVS fragment

```
ExUnd: AXIOM VEXISTS! x: MGod3(x)

re?(x)(w): bool
PossEx: AXIOM VFORALL! x: <> re?(x)

ERC: THEOREM VEXISTS! x: MGod3(x) & re?(x)
```

Here, `re?` is a flexible predicate that corresponds to $E!$ in Eder and Ramharter's notation.

`ExUnd` says that in all worlds, some greatest thing is in "the understanding" (i.e., in the domain of quantification); `PossEx` says that everything exists in reality in some accessible possible world; `ERC` says that in all worlds, there is some greatest thing that exists in reality.

Notice that formalization of these premises in PVS forces attention on the types. Thus, the quantification in `ExUnd` is `VEXISTS` (i.e., modal) rather than `EXISTS` (i.e., classical), which might not have been apparent in Eder and Ramharter's notation, and similarly in `PossEx` and `ERC`.

Greater 5 says that if x does not exists in reality in the actual world and y does exist in reality in some possible world, then the greatness of y in some possible world exceeds that of x in the actual world. Greater 5 uses a similar construction to M-God 3, where quantification over z is used to record the greatness of x in this world for comparison against that of y in some possible world. When formalizing Greater 5 in PVS, it seems sensible to use what we learned with `MGod3` and apply the same technique. That is, we use `MGod3` as a pattern in defining a similar function `M3` that specifies the expression on the right of the implication in Greater 5 as follows.

PVS possibility

```
M3(x, y)(w): bool =
     EXISTS z: (z=g(x) & <> (g(y) > z))(w)
```

Then we can use `M3` in a straightforward specification of Greater 5 such as the following.

PVS possibility

```
Greater5: LEMMA
   VFORALL! x: VFORALL! y: (~re?(x) & <>re?(y) => M3(x, y))
```

But observe that `M3` could equivalently be defined as follows.

PVS possibility

```
M3(x, y): qmlformulas =
  LAMBDA w: EXISTS z: (z=g(x) & <> (g(y) > z))(w)
```

What we will do is employ the body of this definition directly in the specification of `Greater5` given above, thereby removing the need for `M3` (but having benefited from its consideration). This leads to the following PVS specification.

PVS fragment

```
Greater5: AXIOM
   VFORALL! x: VFORALL! y: (~re?(x) & <>re?(y)
     => LAMBDA w: EXISTS z: ((z=g(x) & <> (g(y) > z))(w)))
```

The general technique employed in these examples is to use modal constructions where we can, then "lower" them to classical logic through application to a suitable world when we need to use classical quantification, and to "lift" them back up again with a `LAMBDA` when they are part of a larger modal construction. Observe that this is performed automatically by the `K_conversion` for the subexpressions involving `z`, as it was in `MGod3` (and it would also supply the `LAMBDA w:` had we omitted it).

Some may think it would be better if the conclusion ERC referred to the actual world, rather than (implicitly) all possible words, and likewise ExUnd. It might seem that we could specify this alternative version of ExUnd as follows.

Flawed PVS

```
here: worlds

ExUnd: AXIOM VEXISTS! x: MGod3(x)(here)
```

This is incorrect, however. If we invite PVS to expand out the modal constructs with the command `(grind :if-match nil :exclude "MGod3")`, we obtain the following sequent from this version of `ExUnd`.

PVS sequent

```
  |-------
{1}   EXISTS (x: (vind(w!1))): MGod3(x)(here)
```

We see that PVS has (correctly) required this expression to be true in all worlds, whose Skolemized representative is `w!1`, and this yields an unhelpful constraint on the type for `x`. A correct specification is the following (note the crucial parenthesization around `VEXISTS!`), which causes `x` to have the more useful type `(vind(here))`.

PVS fragment

```
  ExUnd: AXIOM (VEXISTS! x: MGod3(x))(here)
```

Similar care is required with the adjustment to `ERC`.

That concludes our examination of this example. Our interest in it here is purely as an illustration of several tricky topics in the use of quantified modal logic. Those who are interested in the argument itself, and its proof, are referred to [9].

4 Conclusions

PVS is able to embed modal logic in a way the supports its concepts and much of its standard notation while providing the benefits of mechanized checking and automated reasoning. Although the embedding of modal logic is fairly straightforward, checking and automation reveal that use of the quantified logic, in particular, requires care, especially when combined with classical quantification. Thus, readers who have studied and, perhaps, experimented with the examples in the previous subsection may be inclined to agree with Lewis, who writes as follows.

> "Philosophy abounds in troublesome modal arguments—endlessly debated, perennially plausible, perennially suspect. The standards of validity for modal reasoning have long been unclear; they become clear only when we provide a semantic analysis of modal logic by reference to possible worlds and to possible things therein. Thus insofar as we understand modal reasoning at all, we understand it as disguised reasoning about possible beings." [6, p. 175]

Lewis then presents a formal development written directly in terms of possible worlds. In my opinion this goes too far: there is value in the use of modal concepts and notation. However, formalizing modal arguments in PVS can reveal subtleties and unsuspected complexities. In particular, use of conversions and overloading in PVS sometimes produces different interpretations for apparently similar formulas, as in the different readings of the Barcan formula, and it highlights the care needed with the formulas of Section 3.3. In my

view, these should be seen as benefits of formalization and mechanization, not drawbacks: the subtleties and complexities are real, and are exposed by formalization. But, because it can be hard to get the details of modal specifications correct, I highly recommend inviting PVS to expand them out so that (in a compromise with Lewis' position) you can check that their rendition in terms of possible worlds corresponds to your intent.

Benefits of the PVS mechanization of modal logic are illustrated by some published examples concerning the modal ontological argument [10, Section 4], where flawed criticism of a valid modal argument is shown to be due to an erroneous formulation of *modus tollens* (recall Section 2.4), and a first order modal construction encounters some of the difficulties outlined here in Section 3.3.

The embedding technique used here is totally standard (see, for example Wikipedia [1]), but PVS's facilities for overloading, conversions, and binders support it in a particularly attractive way. Embeddings in other verification systems are described by Benzmüller and Woltzenlogel Paleo [2].

Acknowledgments

I am grateful to N. Shankar and Sam Owre, the principal developers of PVS, for much help and advice on both PVS and logic.

References

[1] *Standard Translation (of modal logic into first-order logic).* https://en.wikipedia.org/wiki/Standard_translation.

[2] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Interacting with modal logics in the Coq proof assistant. In *Computer Science—Theory and Applications: 10th International Computer Science Symposium in Russia, CSR 2015*, Volume 9139 of Springer-Verlag *Lecture Notes in Computer Science*, pages 398–411, Springer-Verlag, Listvyanka, Russia, July 2015.

[3] Richard Boulton, Andrew Gordon, Michael Gordon, John Harrison, John Herbert, and John Van Tassel. Experience with embedding hardware description languages in HOL. In *Theorem Provers in Circuit Design (TPCD '92)*, pages 129–156, North Holland, Nijmegen, The Netherlands, 1992.

[4] Günther Eder and Esther Ramharter. Formal reconstructions of St. Anselm's Ontological Argument. *Synthese*, 192(9):2795–2825, October 2015.

[5] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic*, volume 277. Springer Synthese Library, 1998.

[6] David Lewis. Anselm and actuality. *Noûs*, 4(2):175–188, May 1970.

[7] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[8] *PVS home page*. http://pvs.csl.sri.com/.

[9] John Rushby. Mechanized analysis of modal reconstructions of Anselm's traditional Ontological Argument. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, January 2019.

[10] John Rushby. Mechanized analysis of Anselm's modal ontological argument. *International Journal for Philosophy of Religion*, 89:135–152, April 2021. First published online 4 August 2020.