Embedding Modal Logic in PVS

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA

Background for Modal Logic

- The idea is to reason about different modes of truth
  - What it means for something to be possibly true
  - Or to know that something is true
  - As opposed to merely believing it
- The modal qualifiers $\Box$ and $\Diamond$ introduce expressions to be interpreted modally
  - $\Diamond = \neg\Box\neg$, and dually
- All modal logics share basic structure but use different axioms
  - And make other adjustments

  According to the mode attributed to the qualifiers
- For example
  - If $\Box$ is knowledge, we want: $\Box P \supset P$
  - If $\Box$ is belief, instead want: $\Box P \supset \Diamond P$

Simple Example

Uses Alethic modal logic

Where $\Box$ means necessarily true, $\Diamond$ means possibly true

Notation: g is a propositional variable (i.e., 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 (Becker's Postulate)

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

This is actually Hartshorne's rendition of St. Anselm's Modal Ontological Argument for the existence of God (*Proslogion* Chapter III, 1078). It is valid; we'll look at soundness later

History

- Modal reasoning has been studied since Aristotle
- Modern modal logics date to C. I. Lewis, around 1910
  - Propositional modal logic (PML), adds modal qualifiers to classical propositional logic
- Similarly, quantified modal logic (QML) adds qualifiers to first- or higher-order logic
  - Barcan, around 1946
- Semantics in terms of possible worlds due to Kripke, 1959
  - When he was 19 years old

Elementary** Possible Worlds Semantics for PML

- Classical PL is evaluated in some interpretation
  - Assignment of truth values to prop'l variables (i.e., constants)

  Valid sentences (tautologies) are true in all interpretations
- For PML there are multiple worlds (interpretations)
- We lift everything up to become a function `[worlds -> bool]`
- Lifted form of P is $l(P)$, defined recursively on syntax
  - Constants x are lifted by a valuation function V
    - $V(x)(w)$ is value of x in world w
  - Negation: negate the lifted term, $l(\neg P)(w)$ is $\neg l(P)(w)$
  - Conjunction: similarly, $l(P \wedge Q)(w)$ is $l(P)(w) \wedge l(Q)(w)$
    - Other binary connectives are lifted in the same way
  - $l(\Box P)$ is $\forall v : l(P)(v)$**, where v is a fresh variable
  - $l(\Diamond P)$ is $\exists v : l(P)(v)$**, where v is a fresh variable

  Modal sentence P is valid if true in all worlds, $\forall w : l(P)(w)$

Direct Translation of Our Simple Example

Each sentence is translated as the validity of its lifted form

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

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

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

Direct Elementary Translation of Simple Example in PVS

```
direct_hart: 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
%   Remember, PVS universally closes formulas with free variables
  H1: AXIOM EXISTS v: val(g)(v)

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

  HC: THEOREM val(g)(w)
%   Proved by (grind-with-lemmas :polarity? t :lemmas ("H1" "H2"))
END direct_hart
```

Automated Shallow Embedding

- This kind of transformation from one logic or language to another is referred to as a shallow embedding
- It is a syntactic transformation
- So looks like automation needs a syntax-to-syntax translator
- However, capabilities of PVS allow us to do it in PVS itself
- Feasible because the source language, PML, is a logic and has much of its syntax in common with PVS
- Less effective if source were, say, a programming language
- Idea is to define new "modal" operators directly in lifted form
  - e.g., modal conjunction operator `mand` defined as

    ```
    mand(P, Q)(w) = P(w) AND Q(w)
    ```
- PVS allows names to be overloaded (types used to resolve correct instance) so do not need new `mand` just overload `&`
- Which can be used infix (built-in defn is Boolean `AND`)

Elementary PVS Shallow Embedding

```
elem_shallow_pml: THEORY
BEGIN

%   Initial declarations same as xxx

  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) ;

%   Can define other unary and binary connectives similarly

  □(P)(w): bool = FORALL v: P(v) ;
  <>(P)(w): bool = EXISTS v: P(v) ;
%   Or  <>(P): pmlformulas = ~ □ ~ P

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

END elem_shallow_pml
```

Elementary Shallow Embedding of Example in PVS

- PVS has repertoire of unary operators (e.g., $\sim$, $\Box$, and $\Diamond$)
- And infix binary operators (e.g., `&`, `=>`, and `|=`)
- But definitions must use standard prefix `f(x, y)` form

Can now import the embedding and use fairly natural syntax

```
hartshorne1: 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 hartshorne1
```

But what about those ugly appearances of `valid` and `val`?

Neater Shallow Embedding of Example in PVS

- PVS allows functions to be designated as CONVERSIONs
- Applied automatically to subexpressions that would otherwise be type-incorrect
- valid and val as CONVERSIONs fix H1 and H2, but HC needs two conversions
- Define a function validval to do that

Now it looks the way we want it

```
%   These should go in the embedding threory
  validval(x: pvars): bool = valid(val(x))
  CONVERSION valid, val, validval

  H1: AXIOM <> g

  H2: AXIOM P => □ P

  HC: THEOREM g
```

Proof, and Notation Note

- Proof is same as for direct translation, because it expands out to be the same
- After `(lemma "H2") (lemma "H1")`

```
Rule? (grind :if-match nil)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
HC :
{-1}  FORALL w: NOT (FORALL v: NOT val(g)(v))
{-2}  FORALL (P: pmlformulas): FORALL w: P(w) IMPLIES (FORALL v: P(v))
  |-------
{1}   val(g)(w!1)
```

  Observe: this is using the alternative definition of $\Diamond$ in {-1}
  - Not recommended, because harder to interpret
- Note, can use ASCII <>, but [] is preempted in recent PVS
- But those versions allow Unicode, so use hexadecimal 25A1
- LaTeX then needs \usepackage[utf8]{inputenc} \DeclareUnicodeCharacter{25A1}{\ensuremath\Box}

Benefit of the Mechanisms in PVS

Without overloading infix and prefix operators, and conversions

It would look like this

```
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)))

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

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

HC: THEOREM valid(val(g))
```

I think the improvement is obvious

Nonelementary Shallow Embedding

- Suppose we want $\Box$ to mean believes (Doxastic logic)
- Then we want $\Box P \supset \Diamond P$, but not $\Box P \supset P$
- But $\Box P \supset P$ is a theorem of our embedding
- We've inadvertently built too much in
- Our problem is that all worlds are equally accessible
- So $\Box P$ means all worlds, whereas it should mean all worlds accessible from my (current) world
- In the embedding, add a relation `access` and adjust the qualifier rules

```
access: pred[[worlds, worlds]]

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

- Now $\Box P \supset P$ is proveable only if access is reflexive (and v-v)

Accessibility Properties and Standard Axioms

Properties of access relation correspond to standard axioms

T, reflexive: $\Box p \supset p$

4, transitive: $\Box p \supset \Box\Box p$

B, symmetric: $p \supset \Box\Diamond p$

D, serial $(\forall w : \exists v : R(w, v))$**:** $\Box p \supset \Diamond p$

5, Euclidean $(\forall u, v, w : R(u, v) \wedge R(u, w) \supset R(v, w))$**:** $\Diamond p \supset \Box\Diamond p$

Symmetric plus Euclidean is also transitive; reflexive plus Euclidean is also symmetric, hence transitive, hence equivalence

In addition, following are theorems of all modal logics

K: $\Box(p \supset q) \supset (\Box p \supset \Box q)$

N, necessitation: if p is a theorem, so is $\Box p$

Modal Axioms in PVS

- Easy to prove K and N
- It is trivial to prove each of the standard modal axioms follows from its corresponding property of the access relation
- Reverse is much harder
- Generally need to exhibit a counterexample valuation function `val`
- Which means `val` needs to be a variable
- Need right parameterization
- See later

Standard Axioms and Modes

Logics for standard modes are obtained by combining the standard axioms, whose concatenation becomes their name

Alethic, necessity: KT45 (aka S5)

$\Box$ is necessarily and $\Diamond$ is possibly

- The access relation for S5 is an equivalence relation
- S5 has same valid sentences as the elementary treatment (S5 may have several equiv. classes vs. implicitly, one)

Epistemic, knowledge: KT45 (aka S5)

$\Box$ is know and $\Diamond$ is. . . no standard term

Doxastic, belief: KD45

$\Box$ is believe and $\Diamond$ is. . . no standard term

Deontic, duty: KD

$\Box$ is obligated and $\Diamond$ is permited

Temporal, time: KT4 (aka S4)

usually add more structure; LTL and CTL have lots more

Back to the Example

```
hartshorne4: THEORY
BEGIN
  worlds, pvars: TYPE+
  access: pred[[worlds, worlds]]
  val(x:pvars)(w:worlds): bool

  IMPORTING full_shallow_pml[worlds, access, pvars, val]
%           Also provides more_relations, modal_axioms,...
  g: pvars
  P: var pmlformulas

  H1: AXIOM <> g
  H2: AXIOM P => □ P

  HC: THEOREM symmetric?(access) => g
%   Could alternatively cite Modal Axiom B or T5 as premises
END hartshorne4
```

Deep Embedding

- Shallow embedding uses surface syntax
- Deep embedding uses abstract syntax, has structure
- First define a datatype to provide that structure

```
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
```

- Then define validity by recursion and case analysis on this

Validity in Deep Embedding

```
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
  access(w,v): 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: access(w, v) IMPLIES (v |= R)
    ENDCASES
  MEASURE P by <<
  <>(P): modalformula = ~ □ ~ P

  valid(P): bool = FORALL w: w |= P
END deep_pml
```

Deep vs. Shallow Embeddings

- Deep embeddings have advantages when source is a prog. lang.
- Here, shallow embedding is just fine
- But parameterization in deep embedding covenient for proofs that the modal axioms imply properties of the access relation
- Nothing fundamental, just the way I did it
- Recall: need to exhibit counterexample valuation function `val`
- Which means `val` needs to be a variable

Proving Modal Axiom T Entails Reflexivity

```
more_modal_props: THEORY
BEGIN

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

Proof That Modal Axiom T Entails Reflexivity

```
(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)
```

Other axioms are left as exercises

What Does It All Mean?

- Is Hartshorne's argument convincing?
- Much discussion of reasonableness of Modal B in this context
- But, hey, whole thing makes sense only in Alethic logic, KT45
- What about H2? Justified by similarity to N
- But expand them out
- H2: `P!1(w!1) AND access(w!1, v!1) IMPLIES P!1(v!1)`
- N: `(FORALL (w: worlds): P!1(w)) AND access(w!1, v!1) IMPLIES P!1(v!1)`
- Related to fact that deduction theorem (derives H2 from N)
  - `(valid(P) IMPLIES valid(Q)) IMPLIES valid(P => Q)`

  Is not valid in Modal Logic; other direction is OK
- Also, with symmetric access relation, H2 becomes $\Diamond P \supset P$
- So the argument is trivial
- Lesson: modal logic is trickier than you might think

Aside: Strict Implication

- P strictly implies Q
  - $P \dashv\!\!\supset Q$

  If it is not possible for P to be true and Q false
- i.e., in an Alethic modal logic $P \dashv\!\!\supset Q \stackrel{\text{def}}{=} \neg\Diamond(P \wedge \neg Q)$
- It is a theorem that strict implication is the same as necessary material implication:

  $P \dashv\!\!\supset Q = \Box(P \supset Q)$
- 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
- But in an Alethic logic, we have axiom T, so

  $P \dashv\!\!\supset Q \supset P \supset Q$
- But not the converse

Quantified Modal Logic

- Add modal qualifiers to first- or higher-order logic
- Standard step in Anselm's *Proslogion* II argument is to consider "some thing x than which there is nothing greater"
- Modal formulation $\neg\exists y : \Diamond(y > x)$
  - "there is no y greater than x in any (accessible) world"
- Plausible alternative is $\neg\Diamond\exists y : (y > x)$
  - "in no (accessible) world is there a y greater than x"
- Q: Are these the same?
- A: Sometimes they are, and sometimes they are not
- Lesson: quantified modal logic is much trickier than you might think

Elementary Embedding of QML in PVS

- We take the shallow embedding of PML
- Rename `pmlformulas` to `qmlformulas`, and add the following

```
QT: TYPE  % this is the "domain" of quantification
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)
```

- Now let's try to fomulate the Barcan formula in PVS
  - $\forall x : \Box\phi(x) \supset \Box\forall x : \phi(x)$
- Not correct: `CFORALL(□ PP) => □ CFORALL(PP)`
- Correct but ugly:

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

- Apply `K_conversion` (should be called `K_combinator`) as a `CONVERSION` and the first version becomes the second

Continuing. . .

- PVS allows higher-order predicates to be used as binders
- Add ! to name, then `CFORALL!` is a binder, and can write

  `(CFORALL! (s:QT): □ PP(s)) => □ CFORALL! (s:QT): PP(s)`
- Cool, huh? I think this is the way to go
- Now, what is the Barcan formula saying? Look at its contrapositive: $\Diamond\exists x : \phi(x) \supset \exists x : \Diamond\phi(x)$
  - Suppose it's possible that a cow jumped over the moon, then there exists a specific cow that possibly did that
- It says all things that exist in a possible world, also exist in this one
- Considered ontologically offensive (converse, not so)
- But it is a theorem of our embedding
- We've inadvertently built too much in
- "Barcan up the wrong tree," according to Peter

Variable Domains

- We need to allow that not all members of the domain of quantification exist in all worlds

- Introduce higher-order predicate `vind(w)` (values in domain) that identifies the members of `QT` that are defined in world `w`

- Then restrict the quantifiers to just the defined values

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

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

- Note, we are exploiting predicate subtypes here
  Textbooks get into free logics and other complications

Variable Domains and Barcan Formulas

First, characterize how domains may change across access relation

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

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

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

Easy to prove relationships between these and the Barcan formula and its converse

```
%   Requires fixed; would like to say "fixed IMPLIES..."
  vBarcan_eq: LEMMA VFORALL(□ PP) = □ VFORALL(PP)

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

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

Variable Domains and Barcan Formulas (ctd.)

- Other direction, more difficult both to state and to prove
  - Would like to say `Barcan IMPLIES nondecreasing`
- But must state the formulas rather than simply cite their names
- Then, be explicit about scope of quantification of `PP`

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

- 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)
```

Variable Domains and Barcan Formulas (ctd. 2)

- 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

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

- Possible further complication: a value may exist in different worlds, but denote different objects
  - Adds complexity but little expressive value and I omit it
  - 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

Pragmatics

- Russell:

  > 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'

- Guest compares size of yacht in world of his imagination against its size in real world; owner is rooted in real world
- How do we formulate these kinds of comparisons?
- Concrete example
  - Earlier,saw $\neg\exists y : \Diamond(y > x)$ as formulation for "some thing x than which there is nothing greater"
  - Problem is there may be a y greater than x in some worlds, but its greatness in those worlds is exceeded by greatness of x in the actual world

Pragmatics (ctd. 1)

- Eder and Ramharter propose following definition for the predicate G that recognizes maximally great things
- **Def M-God 3:** $Gx :\leftrightarrow \exists z\,(z = g(x) \wedge \neg\Diamond\exists y\,(g(y) \succ z))$

  Here $g(x)$ is the greatness of x and $\succ$ is an ordering (actually an uninterpreted predicate) on greatness.
- 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
- How to write this in PVS? Quantification on `y` is modal (`VEXISTS`) whereas that on `z` seems to be classical (`EXISTS`)
- First, note that greatness is a flexible function: its value depends on the world, so is really $g(x)(w)$
- Whereas $\succ$ is a fixed or rigid predicate: it does not depend on the world

First Attempt at E&R Example

```
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 % flexible function
  >(a, b): bool      % rigid predicate

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

Pretty direct transliteration of $\exists z\,(z = g(x) \wedge \neg\Diamond\exists y\,(g(y) \succ z))$

But not type-correct (inner not a `bool`, outer not a `qmlformulas`)

Corrected E&R Example

- Both issues fixed by dropping down from `qmlformulas` to `bool`

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

  But notice the comparisons `z = g(x)` and `g(y) > z` are not type-correct; they are silently made so by `K_conversion`

- If we do `M-x PPE`, we get the real thing

```
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)
```

- If we break with E&R, can use simpler alternative

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

  But note the difficult `LAMBDA`

Lesson: Quantified Modal Logic is Trickier Than It Looks

- More From E&R Example

  Greater 5: $\forall x \forall y(\neg re(x) \wedge \Diamond re(y) \rightarrow \exists z\,(z = g(x) \wedge \Diamond(g(y) \succ z)))$

- Exercise: do this in PVS (re is flexible)

- "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" D. Lewis

- He presents formalizations directly in terms of possible worlds

- Goes too far: there is value in modal concepts & notation

- However, PVS reveals unsuspected subtleties & complexities

- Compromise: `M-x PPE` and check possible worlds interp'n

Documentation

- Draft Tech Report available
- Feedback welcome
- I plan to make it and the examples publically available soon