Propositional Logic Interpretation of Answered Modal Logic

by Sven Nilsen, 2020

In this paper I show that Answered Modal Logic can be interpreted, in part, with Propositional Logic. This model can only be fully developed under Higher Order Operator Overloading.

For `n` variables, the semantic model of canoncial expressions in Answered Modal Logic form an n-dimensional cube, where each dimension represents a variable in a set of 3 possible states `{□, ◇, ¬◇}`.

The corners of this n-dimensional cubes naturally can model propositions:

□X	X is true
¬◇X	X is false

For example, the propositional logic gate NOT can be defined as following:

not(□X) = ¬◇X	Carries over from Propositional Logic
not(¬◇X) = □X	Carries over from Propositional Logic

However, when trying to extend this operator by imagining an axis flip operation of an interval:

not(◇X) = not((¬◇, □]X) = [¬◇, □)X

This does not work directly, since `[¬◇, □)` is not expressible as a member of the set `{□, ◇, ¬◇}`.

However, `[¬◇, □)` is encodable in the semantic model as:

¬◇	◇	□
1	1	0

This model is extracted from the canonical expression `¬◇X ∨ ◇X`.
This means that `[¬◇, □)X = ¬◇X ∨ ◇X = {¬◇, ◇}X = ¬□X`.
It can be expressed indirectly using two terms `¬◇X ∨ ◇X` instead of a single term.

Therefore:

not(◇X) = ¬□X

Since `not . not <=> id` this gives:

not(¬□X) = ◇X

One the next page, I give a proof of the statement above.

Here is the proof of `not(¬□X) = ◇X`:

∵	not(¬□X)	
∴	not({¬◇, ◇}X)	
∴	not(¬◇X ∨ ◇X)	
∴	not(¬◇X) ∧ not(◇X)	Assuming `∨`[not] <=> `∧`
∴	□X ∧ ¬□X	
∴	□X ∧ {¬◇, ◇}X	
∴	□X ∧ (¬◇X ∨ ◇X)	
∴	(□X ∧ ¬◇X) ∨ (□X ∧ ◇X)	
∴	false ∨ ◇X	
∴	◇X	

In summary, the NOT gate is defined as following (notice how it differs from the `¬` operator):

not(¬◇X) = □X	¬¬◇X = ◇X
not(◇X) = ¬□X	¬◇X = ¬◇X
not(¬□X) = ◇X	¬¬□X = □X
not(□X) = ¬◇X	¬□X = {¬◇, ◇}X

This is also consistent with:

not[¬] <=> not

To derive a full Propositional Logic interpretation, it is sufficent to construct a NAND gate.
Since NOT is already defined, the remaining work is to construct a AND gate:

and(¬◇X, ¬◇X) = ¬◇X	Carries over from Propositional Logic
and(¬◇X, ◇X) = ¬◇X	See proof D (next page)
and(¬◇X, ¬□X) = ¬◇X	See proof A (next page)
and(¬◇X, □X) = ¬◇X	Carries over from Propositional Logic
and(◇X, ¬◇X) = ¬◇X	See proof D (next page)
and(◇X, ◇X) = ◇X	See proof C (next page)
and(◇X, ¬□X) = ◇X	See proof A (next page)
and(◇X, □X) = ◇X	See proof B (next page)
and(¬□X, ¬◇X) = ¬◇X	See proof A (next page)
and(¬□X, ◇X) = ◇X	See proof A (next page)
and(¬□X, ¬□X) = ¬□X	See proof C (next page)
and(¬□X, □X) = ¬□X	See proof B (next page)
and(□X, ¬◇X) = ¬◇X	Carries over from Propositional Logic
and(□X, ◇X) = ◇X	See proof B (next page)
and(□X, ¬□X) = ¬□X	See proof B (next page)
and(□X, □X) = □X	Carries over Propositional Logic

Notice that this is an operator on the functions of a variable `X`,
which is permitted by using Higher Operator Overloading (HOOO) semantics.

The NAND gate is constructed by using `nand <=> not . and`.

For functions of different variables, one can not use the semantics of HOOO.
However, in some cases there exists a model in Propositional Logic.

$\text{and}(\Box X, \Box Y)$ Undefined, but has a model in propositional logic (corners of the cube)
$\text{and}(\Box X, \Diamond Y)$ Undefined, no model

The proofs of AND are as following.

Since `and` is commutative:

$\text{and}(\neg\Diamond X, \neg\Box X) = \text{and}(\neg\Box X, \neg\Diamond X)$
$\text{and}(\Box X, \neg\Box X) = \text{and}(\neg\Box X, \Box X)$

These cases are somewhat intuitive, since `¬□X = {¬◇, ◇}X`:

$\text{and}(\neg\Diamond X, \neg\Box X) = \neg\Diamond X$ Proof A
$\text{and}(\Diamond X, \neg\Box X) = \Diamond X$

The second case is a bit trickier:

$\because$ $\text{and}(\Box X, \neg\Box X)$ Proof B
$\therefore$ $\text{and}(\Box X)(\neg\Box X)$
$\therefore$ $\text{id}(\neg\Box X)$ Using `and(true)(x) => id(x)`
$\therefore$ $\neg\Box X$

The same trick can be used here:

$\text{and}(\Box X, \Diamond X) = \Diamond X$

Two trivial cases are the following:

$\text{and}(\neg\Box X, \neg\Box X) = \neg\Box X$ Proof C
$\text{and}(\Diamond X, \Diamond X) = \Diamond X$

The only case left (two commutative) is the following, which I define using intuition:

$\text{and}(\neg\Diamond X, \Diamond X) = \neg\Diamond X$ Proof D

Q.E.D.