Extracting Bits in Answered Modal Logic

by Sven Nilsen, 2020

In this paper I describe the extraction process of Answered Modal Logic in more detail.

The expression `◇X ∨ □X` can be encoded as following:

011

The first bit tells whether `¬◇` belongs to the set.
The second bit tells whether `◇` belongs to the set.
The third bit tells whether `□` belongs to the set.

By modeling `◇` as an interval:

◇X = (¬◇, □]X

From this one can deduce that the binary representation can be normalized (minimum `1`s):

011 => 010

Hence, the expression can also be normalized further:

◇X ∨ □X => ◇X

When normalizing one can use the following law:

¬□X = {¬◇, ◇}X = ¬◇X ∨ ◇X

It is easy to figure out that `¬□` refers to the whole modal set:

∴ ¬□X = {¬◇, ◇, □}X

∵ ¬□X = ¬◇X ∨ ◇X = ¬◇X ∨ (◇X ∨ □X) = ¬◇X ∨ ◇X ∨ □X = {¬◇, ◇, □}X

When normalizing to minimum `1`s, this leads to:

111 => 110

However, one does not need to choose either normalizing to minimum `1`s or maximum `1`s.

Instead, one can choose the following rules:

011	=>	010	◇X ∨ □X => ◇X
110	=>	111	¬◇X ∨ ◇X => ¬□X

Here is another case:

¬◇X ∨ □X

When a question is *known* **answered** or *known* **unanswered**, it is just *known* **answered**.
It is impossible to unanswer a question, once it is answered.

101 => 001 ¬◇X ∨ □X => □X

The last case is an idea that is impossible to express:

000 ?

This is because an empty expression `` is the same as `¬□X`.
There is no way to write down an expression that corresponds to `000`.

However, it is nice to have 2 codes for each expression, so by definition:

000 => 100 ? => ¬◇X

The intuition of this is when somebody answers a question with gibberish, if the person who asks the question might believe this is the only answer that will be given, then it is known that the question is unanswered. Hence, the question is *known* **unanswered** (`¬◇X`).

Here are all the cases:

000 ¬◇X
001 □X
010 ◇X
011 ◇X
100 ¬◇X
101 □X
110 ¬□X
111 ¬□X

In summary there are 4 normalization rules that gives a symmetric binary encoding:

000 => 100 ? => ¬◇X
011 => 010 ◇X ∨ □X => ◇X
110 => 111 ¬◇X ∨ ◇X => ¬□X
101 => 001 ¬◇X ∨ □X => □X

Normalization of this kind can be performed after or before conversion to binary:

∴ normalize_binary · expression_to_binary <=> expression_to_binary · normalize_expression

∵ normalize_expression : expr → expr
∵ normalize_binary : [bool; 3] → [bool; 3]
∵ expression_to_binary : expr → [bool; 3]