Technical Report: Implementing a Modal Mental Model Reasoner in Python and on the Web

M. Ragni & M. Rocholl

May 7, 2018

1 Introduction

The goal is to implement a modal mReasoner[1] for the Web. Let us first consider an example from sentential reasoning: There exists illusions in reasoning about propositional assertions [**?**]. Illusions are inferences where participants are convinced that the drawn conclusions are true, while they are wrong:

> (1) Consider, for instance, the following assertion: "You have the bread, or else you have the soup or else the salad. Given the further premise: You have the bread." What follows? Can you have the soup too? What about the soup and the salad?

Please note that participants were instructed to interpret the or else as an exclusive or (XOR). Hence the problem can be reformulated as bread XOR soup XOR salad. But, only 17% of the participants gave the correct answer that you can have all three. This answer is predicted by the underlying mental model representation. How do participants process this example?

1.1 System 1: Model generation

Participants first interpret the first connective XOR as follows:

You have the bread or else you have the soup, ...

This leads to an iconic model of the form:

Bread

Soup

This depends on the interpretation of the sentence. This interpretation I is (because of the connective or) a relation R over propositional statements (i.e., atomic propositions with the connectives and ($\wedge$), or ($\vee$), not ($\neg$)) and a discrete twodimensional structure (we choose $\mathbb{N} \times \mathbb{N}$) and use for practical reasons in the implementation as data type a matrix or array. Hence, we define the relation R inductively:

[1] `http://mentalmodels.princeton.edu/models/mreasoner/`

Proposition	Mapping
A and B	$A^I = (x, y)$, $B^I = (x + i, y)$ with $x, y, i \in \mathbb{N}$ and $i > 0$
A xor B	$A^I = (x, y)$, $B^I = (x + i, y - j)$ with $x, y, i \in \mathbb{N}$ and $i, j > 0$
if A, then B	$A^I = (x, y)$ and $B^I = (x + i, y)$ with $x, y, i \in \mathbb{N}$ and $i > 0$
A or B	$A^I = \{(x, y), (x - k, y)\}$, $B^I = (x + i, y + j), (x - k, y + j)\}$ with $x, y, i, j \in \mathbb{N}$ and $i, j, k > 0$

A *minimal mental model* is one that satisifies the constraints from the table with smallest i, j, k.

Definition 1 (Parsing principle) *The parsing and interpretation of a formula takes place in a small scope, i.e., each connective is interpreted successivly and the next proposition is then integrated.*

hence the new information *. . . or else the salad* is processed and the mental model is updated:

Bread

Soup

Salad

hence the mapping is $(1, 1) = Bread^I$, $(2, 2) = Soup^I$, $(3, 3) = Salad^I$. A non-empty cell represents that the respective item is true. In Python the array is initialized with the package numpy[2]. This concludes the model generation phase.

1.2 System 1: Model inspection

Given is Bread and the program checks for the computed model, if Bread and Soup is possible. As there is no row with Bread and Soup the program returns NO. The same holds for all three, the model returns again NO.

1.3 System 2: Flesh out phase

Each empty entry in the array is fleshed out. Hence, the program returns YES.

2 Extension/Formalization for Reasoning with Modals

2.1 The model theory of epistemic possibilities

1. Models represent possibilities (J-L & Byrne, 1991)

[2] `https://stackoverflow.com/questions/6667201/how-to-define-two-dimensional-array-in-python`

2. Compounds of alternatives refer by default to exhaustive conjunctions of possibilities, e.g.: *A or else B, but not both*
has two mental models (system 1):

A
B

(J-L, Khemlani, & Goodwin, 2015.)
Fully explicit models (system 2) also represent what's impossible.

3. A conjunction, *A and B*, makes a factual claim: both propositions hold in all possibilities.

4. Parsimony: *possible that A and possible that B* has a mental model of a single possibility:

A B

2.1.1 Experiment 2

- Parsimony unifies possibilities, e.g.:
Possibly, Tom is here and possibly Ann is here
has mental model of one possibility of who?s here:
Tom Ann
∴ *Possibly, Tom is here and Ann is here.*
(Yes in model theory/ No in modal logics)

- Control problems, e.g., converse inference:

- *Possibly, Tom is here and Ann is here.*
∴ *Possibly Tom is here and possibly Ann is here.* (Yes/Yes)

2.2 Parsing Grammar

The parsing grammar for modal reasoning is specified in the Backus-Naur-Form:

⟨*atom*⟩ ::= ⟨*string_without_spaces_and_tabs*⟩
⟨*necessary*⟩ ::= []
⟨*possibly*⟩ ::= `<>`
⟨*not*⟩ ::= `~`
⟨*and*⟩ ::= `&`
⟨*or*⟩ ::= `or`
⟨*xor*⟩ ::= `xor`
⟨*imply*⟩::= `->`
⟨*biconditional*⟩::= `<->`
⟨*operation*⟩ ::= ⟨*atom*⟩ | ⟨*necessary*⟩ ⟨*operation*⟩ | ⟨*possibly*⟩ ⟨*operation*⟩ |
⟨*not*⟩ ⟨*operation*⟩ | ⟨*operation*⟩⟨*and*⟩⟨*operation*⟩ | ⟨*operation*⟩⟨*or*⟩⟨*operation*⟩

$| \langle operation \rangle \langle xor \rangle \langle operation \rangle | \langle operation \rangle \langle implies \rangle \langle operation \rangle | \langle operation \rangle \langle biconditional \rangle \langle operation \rangle$
$\langle expr \rangle ::= \langle operation \rangle | \langle operation \rangle \langle expr \rangle$

2.3 The process model

Process model of the modal implementation:

Step 1 Expression is parsed by the parsing algorithm based on the grammar introduced above.

Step 2 Convert the list representing the parsed expression into a string that can be interpreted by Pythons SymPy library.

Step 3 The function sympify from the sympy library in Python then processes the expression in a manner defined by the normal logical rules. The rules are here redefined to better represent the "human" understanding.

Step 4 The sympified expression is an object of the logical class that has the least depth. Using the attributes of the logical object, the atoms are cached within it as an attribute, we hence populate an array of the dimension $| Atoms |$. I.e., Xor(Bread, Butter, Milk) $\rightarrow 3D$. Each Atom has one column and each row is one possibility. For each logical operator, different rules apply to the process of populating the model. Step 4 is repeated recursively for each operator in the expression.

Step5 The final matrix is then evaluated using bitmaps of the different rows and columns. Possible inferences are returned.

The parsed expression:

$\langle Expr \rangle ::=$ `Bread xor Butter xor Salad`

yields the array:

['Bread', 'xor', 'Butter', 'xor' 'Salad']

After formatting:

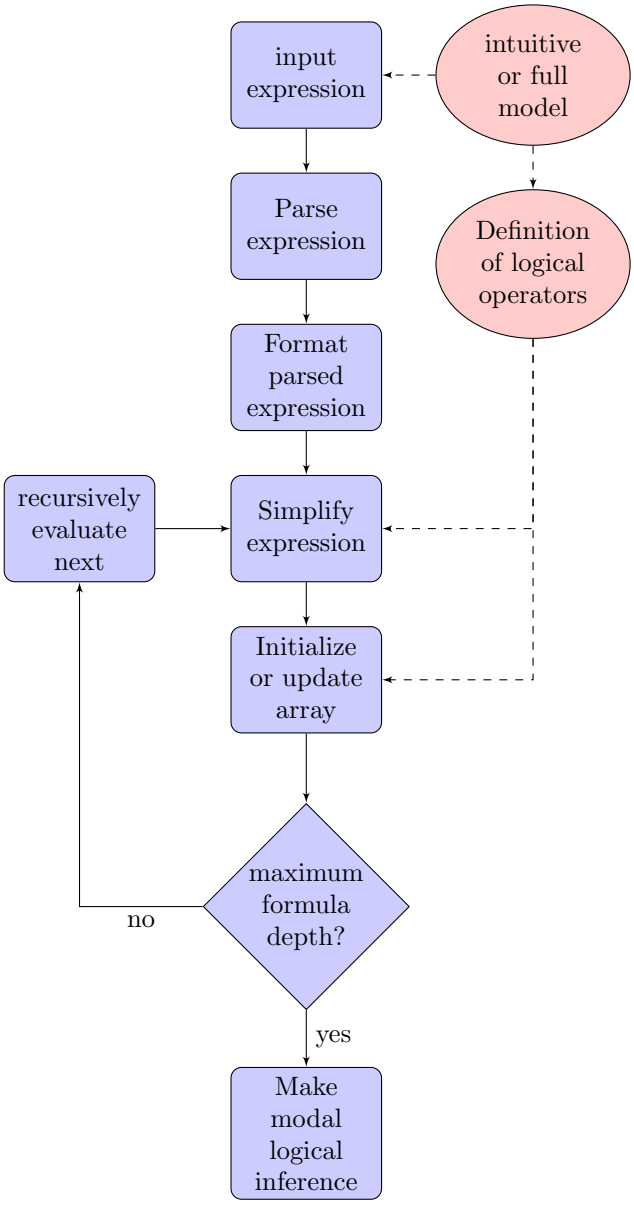
Xor(Bread, Butter, Salad)

After passing the string to the sympify function:

Xor(Bread, Butter, Salad)
Object-type: Xor
$Xor \rightarrow attributes :=$ {Bread, Butter, Salad}

Repeat evaluation with the original expression substituting Xor by Or.

Flow chart representation of algorithm



3 State & Next steps

- The parser already works
- The next step is to model the possibility operator in the model