Reasoning processing

Bachus Naur Form of the modal logic parser

<atom> ::= <string\_without\_spaces\_and\_tabs>

<necessary> ::= “[]”

<possibly> ::= “<>”

<not> ::= “~”

<and> ::= “&”

<or> ::= “|”

<xor> ::= “^”

<imply>::= “->”

<biconditional>::= “<->”

<operation> ::= <atom> | <necessary> <operation> | <possibly> <operation> | <not> <operation> |

<operation><and><operation> | <operation><or><operation> |

<operation> <xor> <operation> | <operation> <implies> <operation> | <operation> <biconditional> <operation>

<expr> ::= <operation> | <operation> <expr>

Process modell of the modal logic reasoning implementation:

Step 1:

Expression is parsed by the parsing algorithm based on the grammar introduced aboce.

Step 2:

Konvertieren des so erhaltenen Arrays in eine für die “Sympy”- Library lesbares -String Format.

Schritt 3:

Evaluieren des so erhaltenen Strings auf Basis der Implementierung der verschiedenen Logischen Operatoren die oben vorhanden sind.

Hierbei wird unterschieden, ob es sich um ein intuitive Evaluation handelt oder ob die “logisch” korrekte Definition der Operatoren unterstellt wird. Je nachdem werden andere intuitive Varianten der logischen Klassen instanziiert.

Schritt 4

Aus dem so erhaltenen logischen Objekt, wird anhand der Argument-Attribute, die durch die logischen Instantiierungen mitgeführt werden, numpy.arrays definiert, die die gleiche Dimensionalität haben wie es Atome in der Expression gibt, Eg. XOR(Bread, Butter, Milk) → 3D.

Das mehrdimensionale Array wird hierbei durch rekursives Auswerten der erhaltenen logischen Formel ausgefüllt.

Dabei werden die einzelnen Modelle jeweils auf einer Reihe dargestellt.

Die Matrix zeigt insgesamt alle mentalen Modelle an, die möglich sind.

Über Bitmaps lässt sich die Matrix dann evaluieren und mögliche Aussagen konstruieren.

Hier ein Beispiel:

The parsed expression “Bread ^ Butter ^ Salad” yields the array:

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

After formatting:

Xor(Bread, Xor(Butter, Salad))

After sympify:

Xor(Bread, Butter, Salad)

Xor.vars → attribute \_attr : frozenset( Bread, Butter, Salad)

Init: np.zeros((len(\_attr), len(\_attr)))

>>> np.array( [0 0 0],

[0 0 0],

[0 0 0] )

Put the atoms on the diagnonal of the np.array (Principle of truth)

>>> Xorarr = np.array( [Bread 0 0 ],

[0 Butter 0 ],

[0 0 Salad ] )

Evaluate the array with Bit-Masks. For instance check if:

np.all(arr[: , 0] == “Bread”) | if True → Necessary

if np.any(…) | if True → Possibly

Afterwards, fill up the rest of the matrix with bitwise operations of the operator in case.

Repeat evaluation with the original expression substituting Xor by Or.

Compare both np.arrays.

>>> ORarr = np.array( [Bread 0 0 ],

[0 Butter 0 ],

[0 0 Salad ],

[Bread Butter Salad ])