```
# -*- coding: UTF-8 -*-

## Data:  24-04-2011

from Expressoes import IMPLIES, AND, OR, NOT, Variavel
from abc import ABCMeta, abstractmethod

class InferenceRule(object):
    ''' Classe abstrata que representa uma regra de inferencia. '''

    __metaclass__ = ABCMeta

    def __init__(self, nome, premissas, conclusao = None):
        ''' premissas :: [Expression], conclusao :: Expression'''
        if isinstance(premissas, list):
            self.premissas = premissas
        else:
            self.premissas = [premissas]

        self.conclusao = conclusao
        self.nome      = nome

    def __str__(self):
        ''' Retorna uma string no modelo:

            [1] primeira premissa
            [.] n-esima premissa
                -----------------
            [Q] conclusao
        '''
        string = ''
        for i, prem in enumerate(self.premissas):
            string += ' [' + str( (i + 1) ) + '] ' + str(prem) + '\n'

        string += '     ' + '-' * 7 + '\n'
        string += ' [Q] ' + str(self.conclusao)
        return string

    @abstractmethod
    def eval(self):
        ''' Se as premissas estiverem bem formadas e nao houver conclusao
        definida, entao este metodo calcula a conclusao e retorna uma refe
        rencia desta R.I. '''
        pass

class AnyInference(InferenceRule):
    ''' Inferencia anonima usada para criar objetos para comparacoes. '''

    def __init__(self, premissas = None, conclusao = None):
        InferenceRule.__init__(self, "AnyInference", premissas, conclusao)

    def eval(self):
        pass
```

```
class ModusPonens(InferenceRule):
    '''
        [1] p -> q
        [2] p
            ------
        [Q] q
    '''

    def __init__(self, premissas = None, conclusao = None):
        ''' p1, p2, conclusao :: Expression'''
        InferenceRule.__init__(self, "Modus Ponens", premissas, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean'''
        if len(other.premissas) != 2:
            return False

        p1        = other.premissas[0]
        p2        = other.premissas[1]
        conclusao = other.conclusao

        if p1 == None and p2 != None and conclusao != None:
                return True

        elif p2 == None and p1 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and conclusao == p1.exp2:
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if isinstance(p1, IMPLIES) and p2 == p1.exp1:
                return True

        else:
            if isinstance(p1, IMPLIES) and (p2 == p1.exp1) \
                and (conclusao == p1.exp2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        self.conclusao = p1.exp2
        return self

class ModusTollens(InferenceRule):
    '''
        [1] p -> q
        [2] !q
            ------
        [Q] !p
    '''

    def __init__(self, premissas = None, conclusao = None):
```

```
        ''' p1, p2, conclusao :: Expression'''
        InferenceRule.__init__(self, "Modus Tollens", premissas, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 2:
            return False

        p1 = other.premissas[0]
        p2 = other.premissas[1]
        conclusao = other.conclusao

        if p1 == None and p2 != None and conclusao != None:
            if (isinstance(p2, NOT) or isinstance(conclusao, NOT)) \
                and p2 != conclusao:
                return True

        elif p2 == None and p1 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and NOT(p1.exp1) == conclusao:
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if isinstance(p1, IMPLIES) and NOT(p1.exp2) == p2:
                return True

        else:
            if isinstance(p1, IMPLIES) and (NOT(p1.exp2) == p2) \
                and (NOT(p1.exp1) == conclusao):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        self.conclusao = NOT(p1.exp1)
        return self

class SilogismoHipotetico(InferenceRule):
    '''
        [1] p -> q
        [2] q -> r
            ------
        [Q] p -> r
    '''

    def __init__(self, premissas = None, conc = None):
        ''' p1, p2, conclusao :: Expression '''
        InferenceRule.__init__(self, "Silogismo Hipotetico", premissas, conc)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 2:
            return False
```

```
        p1 = other.premissas[0]
        p2 = other.premissas[1]
        conclusao = other.conclusao

        if p1 == None and p2 != None and conclusao != None:
            if (isinstance(p2, IMPLIES) and isinstance(conclusao, IMPLIES)) \
                and (p2.exp2 == conclusao.exp2):
                return True

        elif p2 == None and p1 != None and conclusao != None:
            if (isinstance(p1, IMPLIES) and isinstance(conclusao, IMPLIES)) \
                and (p1.exp1 == conclusao.exp1):
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if (isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES)) and \
               (p1.exp2 == p2.exp1):
                return True

        else:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(conclusao, IMPLIES)\
                and (conclusao.exp1 == p1.exp1 and conclusao.exp2 == p2.exp2) \
                and (p2.exp1 == p1.exp2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        self.conclusao = IMPLIES(p1.exp1, p2.exp2)
        return self

class SilogismoDisjuntivo(InferenceRule):
    '''
        [1] p | q
        [2] !p
            -----
        [Q] q
    '''

    def __init__(self, premissas = None, conc = None):
        ''' p1, p2, conclusao :: Expression '''
        InferenceRule.__init__(self, "Silogismo Disjuntivo", premissas, conc)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 2:
            return False

        p1 = other.premissas[0]
        p2 = other.premissas[1]
        conclusao = other.conclusao
```

```
        if p1 == None and p2 != None and conclusao != None:
            if isinstance(p2, NOT) and not isinstance(conclusao, NOT):
                return True

        elif p2 == None and p1 != None and conclusao != None:
            if isinstance(p1, OR) and (conclusao == p1.exp2):
                return True
            elif isinstance(p1, OR) and (conclusao == p1.exp1):
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if isinstance(p1, OR) and (NOT(p1.exp1) == p2):
                return True
            elif isinstance(p1, OR) and (NOT(p1.exp2) == p2):
                return True

        else:
            if isinstance(p1, OR) and \
            ( ((NOT(p1.exp1) == p2) and p1.exp2 == conclusao) or \
              ((NOT(p1.exp2) == p2) and p1.exp1 == conclusao) ):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        if NOT(p1.exp2) == p2:
            self.conclusao = p1.exp1
        else:
            self.conclusao = p1.exp2
        return self

class Adicao(InferenceRule):
    '''
        [1] p
            -----
        [Q] p | q
    '''

    def __init__(self, premissa = None, conclusao = None):
        ''' p1, conclusao :: Expression '''
        InferenceRule.__init__(self, "Adição", premissa, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 1:
            return False

        p1 = other.premissas[0]
        conclusao = other.conclusao

        if p1 == None and isinstance(conclusao, OR):
```

```
            return True

        elif isinstance(conclusao, OR) and (p1 == conclusao.exp1):
                return True

        return False

    def eval(self):
        p1 = self.premissas[0]
        self.conclusao = OR(p1, Variavel('q', None) )
        return self

class Simplificacao(InferenceRule):
    '''
        [1] p & q
            -----
        [Q] p
    '''

    def __init__(self, premissa = None, conclusao = None):
        ''' p1, conclusao :: Expression '''
        InferenceRule.__init__(self, "Simplificação", premissa, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 1:
            return False

        p1 = other.premissas[0]
        conclusao = other.conclusao

        if conclusao == None and (isinstance(p1, AND)):
            return True

        elif isinstance(p1, AND) and \
            ((conclusao == p1.exp1) ^ (conclusao == p1.exp2)):
            return True

        return False

    def eval(self):
        p1 = self.premissas[0]
        self.conclusao = p1.exp1
        return self

    def eval2(self):
        ''' Metodo desta RI que retorna o segundo membro da 1a premissa'''
        p1 = self.premissas[0]
        self.conclusao = p1.exp2
        return self

class Conjuncao(InferenceRule):
    '''
```

```
        [1] p
        [2] q
            -----
        [Q] p & q
    '''

    def __init__(self, premissas = None, conclusao = None):
        ''' p1, p2, conclusao :: Expression '''
        InferenceRule.__init__(self, "Conjunção", premissas, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 2:
            return False

        p1 = other.premissas[0]
        p2 = other.premissas[1]
        conclusao = other.conclusao

        if p1 == None and p2 != None and conclusao != None:
            if isinstance(conclusao, AND) \
                and ((conclusao.exp2 == p2) ^ (conclusao.exp1 == p2)):
                return True

        elif p2 == None and p1 != None and conclusao != None:
            if isinstance(conclusao, AND) and (conclusao.exp1 == p1):
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if p1 != p2:
                return True

        else:
            if isinstance(conclusao, AND) and (conclusao.exp1 == p1) \
                and (conclusao.exp2 == p2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        self.conclusao = AND(p1, p2)
        return self

class Contraposicao(InferenceRule):
    '''
        [1] p -> q
            --------
        [Q] !q -> !p
    '''

    def __init__(self, premissa = None, conclusao = None):
        ''' p1, conclusao :: Expression '''
```

```
        InferenceRule.__init__(self, "Contraposição", premissa, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 1:
            return False

        p1 = other.premissas[0]
        conclusao = other.conclusao

        if p1 == None and isinstance(conclusao, IMPLIES):
            return True

        elif conclusao == None and isinstance(p1, IMPLIES):
            return True

        elif (isinstance(p1, IMPLIES) and isinstance(conclusao, IMPLIES)) \
            and ((NOT(p1.exp1) == conclusao.exp2) \
            and (NOT(p1.exp2) == conclusao.exp1)):
            return True

        return False

    def eval(self):
        p1 = self.premissas[0]
        self.conclusao = IMPLIES(NOT(p1.exp2), NOT(p1.exp1))
        return self

class Resolucao(InferenceRule):
    '''
        [1] p | q
        [2] r | !p
            ------
        [Q] q | r
    '''

    def __init__(self, premissas = None, conclusao = None):
        ''' p1, p2, conclusao '''
        InferenceRule.__init__(self, "Resolução", premissas, conclusao)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 2:
            return False

        p1 = other.premissas[0]
        p2 = other.premissas[1]
        conclusao = other.conclusao

        if p1 == None and p2 != None and conclusao != None:
            if isinstance(p2, OR) and isinstance(conclusao, OR) and \
                 p2.exp1 == conclusao.exp2:
                return True
```

```
        elif p2 == None and p1 != None and conclusao != None:
            if isinstance(p1, OR) and isinstance(conclusao, OR) and \
                p1.exp2 == conclusao.exp1:
                return True

        elif conclusao == None and p2 != None and p1 != None:
            if isinstance(p1, OR) and isinstance(p2, OR) and \
                 NOT(p1.exp1) == p2.exp2:
                return True

        else:
            if isinstance(p1, OR) and isinstance(p2, OR) \
                and isinstance(conclusao, OR) \
                and (NOT(p1.exp1) == p2.exp2) \
                and (p1.exp2 == conclusao.exp1) \
                and (p2.exp1 == conclusao.exp2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        self.conclusao = OR(p1.exp2, p2.exp1)
        return self

class DilemaConstrutivo(InferenceRule):
    '''
        [1] p -> q
        [2] r -> s
        [3] p | r
            ------
        [Q] q | s
    '''

    def __init__(self, premissas = None, conc = None):
        ''' p1, p2, p3, conclusao :: Expression '''
        InferenceRule.__init__(self, "Dilema Construtivo", premissas, conc)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 3:
            return False

        p1 = other.premissas[0]
        p2 = other.premissas[1]
        p3 = other.premissas[2]
        conclusao = other.conclusao

        if p1 == None and p2 != None and p3 != None and conclusao != None:
            if isinstance(p2, IMPLIES) and isinstance(p3, OR) \
                and isinstance(conclusao, OR) and p2.exp1 == p3.exp2 \
                and p2.exp2 == conclusao.exp2:
```

```
                return True

        elif p2 == None and p1 != None and p3 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and isinstance(p3, OR) \
                and isinstance(conclusao, OR) and p1.exp1 == p3.exp1 \
                and p1.exp2 == conclusao.exp1:
                return True

        elif p3 == None and p1 != None and p2 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(conclusao, OR) and p1.exp2 == conclusao.exp1 \
                and p2.exp2 == conclusao.exp2:
                return True

        elif conclusao == None and p1 != None and p2 != None and p3 != None:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(p3, OR) and p1.exp1 == p3.exp1 \
                and p2.exp1 == p3.exp2:
                return True

        else:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(p3, OR) and isinstance(conclusao, OR) \
                and (p1.exp1 == p3.exp1) and (p2.exp1 == p3.exp2) \
                and (p1.exp2 == conclusao.exp1) and (p2.exp2 == conclusao.exp2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        self.conclusao = OR(p1.exp2, p2.exp2)
        return self

class DilemaDestrutivo(InferenceRule):
    '''
        [1] p -> q
        [2] r -> s
        [3] !q | !s
            ------
        [Q] !p | !r
    '''

    def __init__(self, premissas = None, conc = None):
        ''' p1, p2, p3, conclusao :: Expression'''
        InferenceRule.__init__(self, "Dilema Destrutivo", premissas, conc)

    def __eq__(self, other):
        ''' other :: InferenceRule -> Boolean '''
        if len(other.premissas) != 3:
            return False

        p1 = other.premissas[0]
```

```
        p2 = other.premissas[1]
        p3 = other.premissas[2]
        conclusao = other.conclusao

        if p1 == None and p2 != None and p3 != None and conclusao != None:
            if isinstance(p2, IMPLIES) and isinstance(p3, OR) \
                and isinstance(conclusao, OR) \
                and (NOT(p2.exp1) == conclusao.exp2) \
                and (NOT(p2.exp2) == p3.exp2):
                return True

        elif p2 == None and p1 != None and p3 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and isinstance(p3, OR) \
                and isinstance(conclusao, OR) \
                and (NOT(p1.exp1) == conclusao.exp1) \
                and (NOT(p1.exp2) == p3.exp1):
                return True

        elif p3 == None and p2 != None and p1 != None and conclusao != None:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(conclusao, OR) \
                and (NOT(p1.exp1) == conclusao.exp1) \
                and (NOT(p2.exp1) == conclusao.exp2):
                return True

        elif conclusao == None and p2 != None and p3 != None and p1 != None:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(p3, OR) and (NOT(p1.exp2) == p3.exp1) \
                and (NOT(p2.exp2) == p3.exp2):
                return True

        else:
            if isinstance(p1, IMPLIES) and isinstance(p2, IMPLIES) \
                and isinstance(p3, OR) and isinstance(conclusao, OR) \
                and (NOT(p1.exp1) == conclusao.exp1) \
                and (NOT(p1.exp2) == p3.exp1) \
                and (NOT(p2.exp1) == conclusao.exp2) \
                and (NOT(p2.exp2) == p3.exp2):
                return True
        return False

    def eval(self):
        p1 = self.premissas[0]
        p2 = self.premissas[1]
        self.conclusao = OR( NOT(p1.exp1), NOT(p2.exp1) )
        return self
```