Tutorial-4-Using_the_Boolean_approach_on_your_model

February 1, 2019

The function `your_model` returns all possible response-combinations from social-ecological model.

```
In [51]: from your_model import your_model

         responsesList, niceNames, collectedResponses = your_model()
```

The model concerns a set of management interventions, and their effects on 5 social and ecological response variables.

```
In [52]: # print
         for v in niceNames.values():
             print(v)
```

```
intervention
environmental flow
leisure use of river
water price
satisfaction with water authority
engagement of stakeholders
```

The response combinations have been encoded as follows:

```
In [53]: for k, v in niceNames.items():
             print(k + ': ' + v)
```

```
int: intervention
qua: environmental flow
lei: leisure use of river
pri: water price
sat: satisfaction with water authority
eng: engagement of stakeholders
```

And the response-combinations that were found in the model were returned as follows:

```
In [54]: # print header
         header = [ r[-3:] for r in responsesList ]
         print('  '.join(header))
         print(''.join(['-----']*len(header)))

         # print response combinations found
         for response in collectedResponses:
             print('  '.join(response))
```

```
eng  sat  pri  lei  qua
-------------------------
neg  neg  neg  neg  neg
neg  neg  neg  neg  pos
neg  neg  neg  pos  pos
neg  neg  pos  neg  neg
neg  neg  pos  neg  pos
neg  neg  pos  pos  pos
neg  pos  neg  neg  neg
neg  pos  neg  neg  pos
neg  pos  neg  pos  pos
pos  neg  neg  neg  pos
pos  neg  neg  pos  pos
pos  pos  neg  neg  neg
pos  pos  neg  neg  pos
pos  pos  neg  pos  pos
pos  pos  pos  neg  pos
pos  pos  pos  pos  pos
```

The first step is to assign one of the responses to `True` and the other to `False`.

```
In [55]: str4true = 'pos'; str4false = 'neg'
```

Now we can treat the responses as Boolean variables. We use PyEDA to encode them as Boolean variables.

```
In [56]: from pyeda.inter import espresso_exprs
         from findpcu import getUnobservedInts, getRespvarList2BoolvarList, intList2boolexpr, bo

         x, x2s, r2idx = getRespvarList2BoolvarList(responsesList, str4true, str4false)

         # print the Boolean variable names
         x
```

```
Out[56]: [int_eng, int_sat, int_pri, int_lei, int_qua]
```

```
In [57]: type(x[0])
```

```
Out[57]: pyeda.boolalg.expr.Variable
```

We turn the list of observed responses into a list of unobserved responses (i.e. impossible response combinations), encoded as integers.

```
In [58]: observedInts = [ int(''.join(['1' if i in str4true else '0' for i in responseCombinatio
                                      for responseCombination in collectedResponses ]
         observedInts
         unobservedInts = set(range(2**len(responsesList)))
         # ... and loop through the observed response combinations, discarding those that were o

         for i in observedInts:
             unobservedInts.discard(i)

         unobservedInts
```

```
Out[58]: {2, 6, 10, 12, 13, 14, 15, 16, 18, 20, 21, 22, 23, 26, 28, 30}
```

Using the Boolean variables above, we can create a Boolean expression from the unobserved responses.

```
In [59]: unobservedBoolexpr = intList2boolexpr(unobservedInts, x)

         # print the Boolean expression of unobserved responses
         unobservedBoolexpr
```

```
Out[59]: Or(And(~int_eng, ~int_sat, ~int_pri, int_lei, ~int_qua),
     And(~int_eng, ~int_sat, int_pri, int_lei, ~int_qua), And(~int_eng, int_sat, ~int_pri, int_le
     And(~int_eng, int_sat, int_pri, ~int_lei, ~int_qua), And(~int_eng, int_sat, int_pri, ~int_le
     And(~int_eng, int_sat, int_pri, int_lei, ~int_qua), And(~int_eng, int_sat, int_pri, int_lei,
     And(int_eng, ~int_sat, ~int_pri, ~int_lei, ~int_qua), And(int_eng, ~int_sat, ~int_pri, int_l
     And(int_eng, ~int_sat, int_pri, ~int_lei, ~int_qua), And(int_eng, ~int_sat, int_pri, ~int_le
     And(int_eng, ~int_sat, int_pri, int_lei, ~int_qua), And(int_eng, ~int_sat, int_pri, int_lei,
     And(int_eng, int_sat, ~int_pri, int_lei, ~int_qua), And(int_eng, int_sat, int_pri, ~int_lei,
     And(int_eng, int_sat, int_pri, int_lei, ~int_qua))
```

The complexity of this expression can be reduced using Boolean minimisation, using the espresso algorithm from PyEDA

```
In [60]: boolExprMin, = espresso_exprs(unobservedBoolexpr)

         # print minimised Boolean expression
         boolExprMin
```

```
Out[60]: Or(And(~int_eng, int_sat, int_pri), And(int_sat, int_pri, ~int_qua), And(int_lei, ~int_
```

A more human-readable form of the minimised Boolean expression can be obtained using boolexpr2RespvalList

```
In [61]: PCUList = boolexpr2RespvalList(boolExprMin, x2s)
         PCUList
```

```
Out[61]: [['negint_qua', 'posint_lei'],
          ['posint_eng', 'negint_qua', 'negint_sat'],
          ['posint_sat', 'posint_pri', 'negint_qua'],
          ['posint_sat', 'posint_pri', 'negint_eng'],
          ['posint_eng', 'posint_pri', 'negint_sat']]
```

The function draw_implication_network2 can be used to create an implication network. Here, we have specified that the effects of management interventions on community engagemend, environmental flow, and water price, should be antecedents in the network.

```
In [62]: draw_implication_network2(PCUList,
                                    ['posint_eng', 'negint_eng', 'negint_qua', 'posint_qua', 'pos
                                    'your_model', niceNames = niceNames, controlSymbol = '&#10148
```

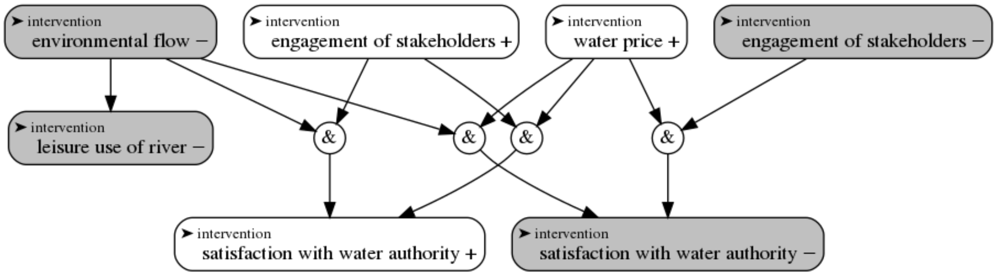
```
your_model.pdf has been created
```

A pdf of the implication network has been created. However we can also use graphviz to create figures in other formats.

```
In [63]: from IPython.display import Image
         import os

         os.system("dot -Tpng your_model.dot > your_model.png")
```

```
Out[63]: 0
```



An implication network for your_model.