Supplementary Information

Dealing with high uncertainty in qualitative network models using Boolean analysis

Kristensen, et al. (2019)

Table of Contents

A An example showing the links between weighted-predictions matrix analysis, loop analysis, and Boolean analysis

The purpose of this appendix is to use the example network in Fig. A.5 to show the relationship between the weighted-predictions matrix (Dambacher et al., 2002), loop analysis (Lane and Levins, 1977), and Boolean analysis. According to the weighted-predictions matrix analysis, the species responses of interest are maximally ambiguous (A.1), however loop analysis shows that the species responses are nevertheless fully determined by the relationships between them (A.2). Boolean analysis uncovers these same deterministic relationships (A.3), and we show how to perform a Boolean minimisation by hand. The tutorial in SI C.1 shows how the example would be implemented in Python.

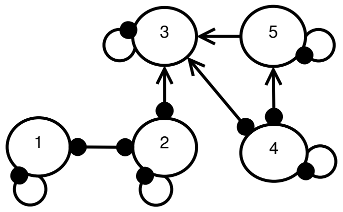


Figure A.5: The ecological interaction network (from Melbourne-Thomas et al., 2012) used for the example in this appendix. Nodes represent species, direct positive effects between species are indicated by an arrow, and negative effects by a filled circle.

A.1 Weighted-predictions matrix analysis

The weighted-predictions matrix W provides a summary of the positive and negative effects between species in a network in a press-perturbation scenario (Dambacher et al., 2002). The elements of the weighted-predictions matrix W_{ij} count the proportion of feedback cycles that conform to the sign given by the adjoint; therefore $W_{ij} = 1$ means that Species i will always respond to a negative press-perturbation of Species j in the direction given by the sign of the corresponding element of the adjoint, and $W_{ij} < 1$ indicates that a mixture of positive and negative feedback cycles are present and so the response direction is ambiguous.

Following Dambacher et al. (2002) and using the same notation, the weighted-predictions matrix for the interaction network in Fig. A.5 is found as follows.

From the community matrix A we obtain the qualitative community matrix $^\circ A$ such that:

$$^\circ A_{ij} = \begin{cases} 0 & \text{if } A_{ij} = 0 \\ -1 & \text{if } A_{ij} < 0 \\ +1 & \text{if } A_{ij} > 0. \end{cases} \tag{A.2}$$

For the network in Fig. A.5, the qualitative community matrix is

$$^{\circ}A = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 \\ 0 & +1 & -1 & +1 & +1 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & +1 & -1 \end{bmatrix} \tag{A.3}$$

By taking the classical adjoint or adjugate of the qualitative community matrix, $\text{adj}(-^{\circ}A)$, an analogue of the sensitivity matrix is obtained

$$-^{\circ}A^{-1} = \frac{\text{adj}(-^{\circ}A)}{\det(-^{\circ}A)} \tag{A.4}$$

where det() is the determinant, and adj() is the classical adjoint or adjugate. The determinant scales the magnitude of response to the press perturbation, and if the system is stable then $\det(-^{\circ}A)$ is positive, while the adjugate gives the sum of feedback cycles that contribute to that species' positive or negative response. For the network in Fig. A.5, the adjoint is

$$\text{adj}(-^{\circ}A) = \begin{bmatrix} +6 & -4 & +2 & +2 & 0 \\ -4 & +4 & -2 & -2 & 0 \\ -2 & +2 & 0 & 0 & 0 \\ +1 & -1 & 0 & +1 & -1 \\ +1 & -1 & 0 & +1 & +1 \end{bmatrix} \tag{A.5}$$

Each i, j element of the adjoint is the sum the positive feedback cycles minus the sum of the negative feedback cycles from j to i. In order to interpret its value as a proportion, one must also find the absolute number of positive and negative feedback cycles in total.

From the community matrix A we also obtain the absolute feedback matrix T, which counts the total number of cycles. This is done in two steps. First, we find the binary community matrix $^{\bullet}A$ such that:

$$^{\bullet}A_{ij} = \begin{cases} 0 & \text{if } A_{ij} = 0 \\ +1 & \text{if } A_{ij} \neq 0. \end{cases} \tag{A.6}$$

Then the absolute feedback matrix T can be found

$$T_{ji} = \text{per}(\min {}^{\bullet}A_{ij}), \tag{A.7}$$

where 'per' is the matrix permanent, and where $\min {}^{\bullet}A_{ij}$ is the submatrix found by removing row i and column j of ${}^{\bullet}A$. For the network in Fig. A.5, the absolute feedback matrix is

$$T = \begin{bmatrix} 6 & 4 & 2 & 2 & 2 \\ 4 & 4 & 2 & 2 & 2 \\ 2 & 2 & 4 & 4 & 4 \\ 1 & 1 & 2 & 3 & 5 \\ 1 & 1 & 2 & 3 & 5 \end{bmatrix} \tag{A.8}$$

Finally the weighted-predictions matrix can be found

$$W = \text{abs}(\text{adj}(-{}^{\circ}A))./T \tag{A.9}$$

where ./ indicates element-wise division, and by convention W_{ij} is set to 1 when $T_{ij} = 0$. For the network in Fig. A.5, the weighted-predictions matrix is

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1/3 & 1/5 \\ 1 & 1 & 0 & 1/3 & 1/5 \end{bmatrix} \tag{A.10}$$

If all feedback cycles are given equal weighting, then the elements of the weighted-predictions matrix can be interpreted probabilistically, and the magnitude of the W_{ij} elements can be interpreted as the degree of evidential support for the response-sign indicated by the adjoint (e.g. Metcalf et al., 2011; Ramos-Jiliberto et al., 2012). According to the weighted-predictions matrix in Eq. A.10, the effects of a press-perturbation of Species 3 upon Species 3, 4, and 5 are maximally ambiguous ($W_{3,3} = W_{4,3} = W_{5,3} = 0$), which is because the number of positive and negative feedbacks is equal. Therefore a probabilistic interpretation would assign a 50% probability of each outcome being either positive or negative.

A.2 Loop analysis

It is possible to obtain information about the species responses by analysing the adjoint of the community matrix directly (c.f. Lane and Levins (1977, p. 464), and Dambacher et al. (2003, p. 82)). Recall that the sensitivity matrix $S = (-A)^{-1}$. Given that $(-A)^{-1} = \text{adj}(-A)/\det(-A)$, and given that in a stable system $\det(-A)$ is positive, then the signs of the elements of $\text{adj}(-A)$ are equal to the signs of the elements of S.

According to the weighted-predictions matrix analysis above, the effects of a press-perturbation of Species 3 upon Species 3, 4, and 5 are maximally ambiguous; however direct analysis reveals that they share a simple relationship. The elements of interest in the adjoint are

$$\mathrm{adj}(-A)_{3,3} = -(a_{45}a_{54} + a_{44}a_{55})(a_{12}a_{21} - a_{11}a_{22}), \tag{A.11a}$$
$$\mathrm{adj}(-A)_{4,3} = a_{43}a_{55}(a_{12}a_{21} - a_{11}a_{22}), \tag{A.11b}$$
$$\mathrm{adj}(-A)_{5,3} = a_{43}a_{54}(a_{12}a_{21} - a_{11}a_{22}). \tag{A.11c}$$

From this, a relationship between the signs of the elements of the community matrix can be obtained

$$\mathrm{sign}(S_{3,3}) = -\mathrm{sign}(S_{4,3}) = -\mathrm{sign}(S_{5,3}). \tag{A.12}$$

In contrast to the weighted-predictions matrix analysis, Eq. A.12 implies that the species responses are not so much 'ambiguous' as contingent upon one another. The relationships between the species responses can be expressed as logical implication statements, for example *if Species 3 responds positively then Species 4 responds negatively.*

A.3 Boolean analysis

The logical relationship between species responses revealed by direct analysis above can also be found using Boolean analysis. Let us assume that a large-enough sampling of the parameter space of the community matrix is performed so that *all* possible species-response combinations in Species 3, 4, and 5 to the negative perturbation of Species 3 are obtained. For simplicity, let us assume that the responses of interest are always positive or negative. Then, because the species-responses are dichotomous, the species-response combinations can be encoded as a truth table, as shown in Table A.3.

Table A.3: A truth table listing all possible species response-combinations, and whether they were never observed or observed. The values '0' correspond to 'false' and '1' to 'true'.

	positive response to negative perturbation of spp 3			
ID	spp 3	spp 4	spp 5	never observed
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

The first step take is to reduce the length of the table by focusing only on those response-combinations that were never observed, resulting in Table A.4.

Table A.4: After reducing the truth table to only those species response-combinations that were never observed, then two rows are selected that can be simplified using Boolean algebra.

	positive response to negative perturbation of spp 3			never
ID	spp 3	spp 4	spp 5	observed
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

The two rows highlighted in Table A.4 can be simplified using Boolean algebra as follows

$$(\text{spp }3\wedge\text{spp }4\wedge\overline{\text{spp }5})\vee(\text{spp }3\wedge\text{spp }4\wedge\text{spp }5)=\text{spp }3\wedge\text{spp }4\wedge(\overline{\text{spp }5}\vee\text{spp }5)=\text{spp }3\wedge\text{spp }4 \tag{A.13}$$

where we have used the standard notation of a bar to indicate 'false' (a negative species response), and no bar to indicate 'true' (a positive species response). The result of the simplification is given in Table A.5, where a '-' is used to indicate species responses that can take either truth value.

Table A.5: Combining rows 6 and 7 gives a new final row in the truth table, where '-' indicates that the entry can take either truth value.

	positive response to negative perturbation of spp 3			never
ID	spp 3	spp 4	spp 5	observed
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
5	1	0	1	1
	1	1	-	1

In a similar way, one can simplify rows 1 and 5 of Table A.5, and then rows 0 and 2 of the resulting table, until one obtains the truth table in Table A.6 that cannot be simplified further.

A rows of the minimised truth table give the 'PCUs' (short for the French 'projection canonique ultime' or 'ultimate canonical projection'), which in this context are the minimal sets describing the never-observed species-response combinations (Theuns, 1994, Section 28.2). From the PCUs, the entire original truth table and all species-response combinations can

Table A.6: The final truth table.

ID	positive response to negative perturbation of spp 3			never observed
	spp 3	spp 4	spp 5	
	0	-	0	1
	-	0	1	1
	1	1	-	1

be recovered. The three rows in Table A.6 describe three sets of never-observed species responses:

1. $\{\overline{\text{spp 3}}, \overline{\text{spp 5}}\}$,
2. $\{\overline{\text{spp 4}}, \text{spp 5}\}$, and
3. $\{\text{spp 3}, \text{spp 4}\}$.

These three never-observed responses have the corresponding meanings, "we never observe ...":

1. "... a decrease in Species 3 and a decrease in Species 5",
2. "... a decrease in Species 4 and an increase in Species 5", and
3. "... an increase in Species 3 and an increase in Species 4".

The truth table in Table A.6 is also the minimal representation of the original table, and the three sets of never-observed species responses are the minimal description of all species response relationships in the system. This table is minimal due to the order in which the simplifications above were performed, however in general a technique like the Quinne-McCluskey method is needed to perform the operations in the right order to get the minimal table.

When the description is minimal, these sets of never-observed species responses are known as the 'PCUs' (Theuns, 1994). Each PCU of length k can be turned into $2^k - 2$ logical implications about response-combinations that were observed:

1. $\overline{\text{spp 3}} \rightarrow \text{spp 5}$, $\overline{\text{spp 5}} \rightarrow \text{spp 5}$;
2. $\overline{\text{spp 4}} \rightarrow \overline{\text{spp 5}}$, $\text{spp 5} \rightarrow \text{spp 4}$; and
3. $\text{spp 3} \rightarrow \overline{\text{spp 4}}$, $\text{spp 4} \rightarrow \overline{\text{spp 3}}$,

with the corresponding meanings:

1. "a decrease in Species 3 implies an increase in Species 5", "a decrease in Species 5 implies an increase in Species 5";
2. "a decrease in Species 4 implies a decrease in Species 5", "an increase in Species 5 implies an increase in Species 4"; and

3. “an increase in Species 3 implies a decrease in Species 4”, “an increase in Species 4 implies a decrease in Species 3”.

By selecting one logical implication statement from each PCU, and by linking statements with shared terms where possible, all species-response combinations can be concisely described by a network of logical implications (Fig. 2d). The minimal table and the logical implications network derived from it holds all of the information needed to reconstruct a complete description of the species-response relationships.

B The community matrix as a derivative description of a Lotka-Volterra system

The purpose of this appendix is to show how the community matrix is a derivative description of the population dynamics compared to the Lotka-Volterra description, and also to describe some of the special properties of the community matrix.

B.1 Background: primary versus derivative properties and PoI

Mikkelson (2004) presents a philosophical argument that PoI should be applied to the primary description of a problem. The distinction between "primary" and "derivative" is that two things cannot differ with respect to their derivative properties without also differing with respect to their primary properties. For example, if two coins are flipped, then the property space may be described as $D_1 = \{TT; HT; TH; HH\}$ or $D_2 = \{2T; 1H, 1T; 2H\}$. Outcomes cannot differ in D_2 without also differing in D_1, therefore D_1 is the more primary description. When PoI is applied to D_1 it leads to the correct (observed) probabilities.

B.2 Showing that the community matrix is a derivative description

The community matrix is derived from the Lotka-Volterra description as follows. Eq. 1 has a steady-state at

$$\mathbf{n}^* = B^{-1}(-\mathbf{r}), \tag{B.14}$$

and a community matrix (also known as the Jacobian matrix)

$$J = \left[\left. \frac{\partial \dot{n}_i}{\partial n_j} \right|_{\mathbf{n}^*} \right] = N^* B \tag{B.15}$$

where N^* is the diagonal matrix formed from $\mathbf{n}^*$. It is assumed that all $n_i^* > 0$, therefore the sign structure in J is the same as the sign structure in B i.e. determined by the signs of the direct interactions in the interaction network (e.g. Fig. 1a). To randomly sample parameter values, a positive scaling factor may be chosen and multiplied by J in Eq. B.15 so that the magnitude of all non-zero elements of J are between 0 and 1.

Another common way to define the community matrix (e.g. Appendices Melbourne-Thomas et al., 2012) is to scale the population-size state variables first

$$\hat{n}_i = \frac{n_i}{n_i^*}, \tag{B.16}$$

so that the steady state

$$\hat{\mathbf{n}}^* = \mathbf{1}. \tag{B.17}$$

Then

$$\frac{d\hat{n}_i}{dt} = \hat{n}_i \left(r_i + \sum_j b_{ij} \hat{n}_i n_j^* \right), \tag{B.18}$$

and the community matrix

$$A = \left[\left. \frac{\partial \dot{\hat{n}}_i}{\partial \hat{n}_j} \right|_{\hat{\mathbf{n}}^* = [1]} \right] = BN^* = (N^*)^{-1} J N^*. \tag{B.19}$$

Again a scaling can be applied so that all elements of A are between 0 and 1.

The community matrix A has three special properties that make it useful for Monte Carlo simulations and predicting species responses to pest control. First, because it is assumed that all species have a positive steady state $n_i^* > 0$, then it preserves the sign structure of J, which is ultimately determined by the signs of the direct interactions. Second, Eq. B.19 is a similarity transform, therefore the eigenvalues (but not eigenvectors) of A and J are equal, and stability can be determined from either. Third, the sensitivity matrix $S = -A^{-1}$ has the same sign structure as $-J^{-1}$, therefore the signs of the species responses determined from J are preserved.

The Lotka-Volterra description is more primary than the community matrix because many B-and-N^* pairs can satisfy Eq. B.15 or B.19 to give the same J or A. Recall that the way to distinguish between primary and derivative properties is: two things cannot differ with respect to their derivative properties without also differing with respect to their primary properties. Each A represents a family of B-and-N pairs, therefore systems can differ in their Lotka-Volterra descriptions yet have the same community matrix description, however two systems with different community matrix descriptions cannot have the same Lotka-Volterra description.

C Tutorials

The following tutorials are available from the Github repository as Jupyter notebooks: https://github.com/nadiahpk/qualitative-modelling/tree/master/tutorials

C.1 Implementing the small example from Fig. 2

The purpose of this tutorial is to show how the weighted-predictions matrix approach, Monte Carlo simulations, and the Boolean approach, would be implemented for the small example from Fig. 2. SI A also shows the theoretical links between loop-analysis, the weighted-predictions matrix, and the Boolean approach.

The weighted-predictions matrix was found using SageMath in Jupyter, and the Notebook is archived as: Tutorial-1.1-Weighted_predictions_matrix_approach.ipynb

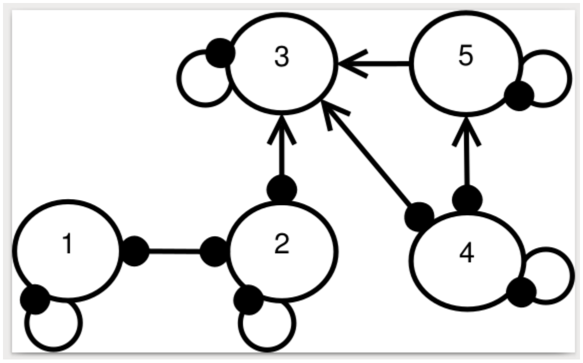
The Monte Carlo simulations and Boolean analysis were performed in Python3 in Jupyter, and the Notebook is archived as:
Tutorial-1.2-Probabilistic_and_Boolean_approaches.ipynb

C.1.1 Finding the weighted-predictions matrix

Weighted-predictions matrix

The weighted-predictions matrix, W, introduced by Dambacher et al. (2002), summarises the positive and negative effects of a press-perturbation on species in an interaction network. In this tutorial, we obtain W for the species interaction network below (`fivevariable2.png`), and use it to predict the effects of a negative press-perturbation of species 3 on species 3, 4, and 5 in the network below. This example is also used in Fig. 2 in the main test of the paper and Supplement A.

```
In [1]: from IPython.display import IFrame
        from IPython.display import Image
        import os

        # display the interaction network used in the example
        #IFrame("fivevariable2.pdf", width=500, height=400)
```



fivevariable2.png

Obtain the weighted-predictions matrix

Step 1: Create the qualitative community matrix

It is convenient to start by encoding the interaction network in a flexible way. Below, we specify:

- spp_list, a list of species names;
- positive_edges_dict, a dictionary that specifies the positive interactions between species; and
- negative_edges_dict, a dictionary that specifies the negative interactions between species.

```
In [2]: #
        spp_list = ['s1','s2','s3','s4','s5']

        # key is recipient of a positive effect
        positive_edges_dict = {
        's5': ['s4'],
        's3': ['s2', 's4', 's5'],
        }

        # key is recipient of a negative effect
        negative_edges_dict = {
        's5': ['s5'],
        's4': ['s4', 's5', 's3'],
        's3': ['s3'],
        's2': ['s2', 's1', 's3'],
        's1': ['s1', 's2'],
        }
```

Because we will be working with matrices, the first step is to assign each species name to an index in the matrix.

```
In [3]: sz = len(spp_list) # the number of species
```

```
        # a dictionary that maps from a species' name to its index in the
        # matrix
        spp2idx = { spp_name: idx
                    for idx, spp_name in enumerate(spp_list) }
        spp2idx
```

```
Out[3]: {'s1': 0, 's2': 1, 's3': 2, 's4': 3, 's5': 4}
```

The qualitative community matrix, denoted $^{\circ}A$ in Dambacher et al. (2002), has a `-1` for negative species interactions, `+1` for positive interactions, and `0` otherwise. So we use the edges dictionaries together with the `spp2idx` dictionary to populate the qualitative community matrix, `Aq`, with zeros, ones, and negative ones accordingly.

```
In [4]: Aq = matrix(QQ, sz, sz)


        for recipient, giverList in positive_edges_dict.items():
            for giver in giverList:
                Aq[ spp2idx[recipient], spp2idx[giver]] = 1

        for recipient, giverList in negative_edges_dict.items():
            for giver in giverList:
                Aq[ spp2idx[recipient], spp2idx[giver]] = -1

        Aq
```

```
Out[4]: [-1 -1  0  0  0]
        [-1 -1 -1  0  0]
        [ 0  1 -1  1  1]
        [ 0  0 -1 -1 -1]
        [ 0  0  0  1 -1]
```

Step 2: Find the adjoint of the qualitative community matrix

The classical adjoint or adjugate of the qualitative community matrix, $\mathrm{adj}(-^{\circ}A)$, provides an analogue of the sensitivity matrix

$$-^{\circ}A^{-1} = \frac{\mathrm{adj}(-^{\circ}A)}{\det(-^{\circ}A)}$$

where det() is the determinant, and adj() is the classical adjoint or adjugate. It equally weights each feedback loop between species in the system (magnitude = 1 for all), and counts the sum of those positive (+1) and negative (-1) feedbacks.

```
In [5]: adj = (-Aq).adjoint()
        adj
```

```
Out[5]: [ 6 -4  2  2  0]
        [-4  4 -2 -2  0]
        [-2  2  0  0  0]
        [ 1 -1  0  1 -1]
        [ 1 -1  0  1  1]
```

Because the elements in adj$(-{}^{\circ}A)$ are the sum of positive and negative feedbacks, their signs are indicative of the response of each row-species to a negative press-perturbation of each column-species. For example (taken from Dambacher et al. (2002)), if there are 4 positive feedback loops between two species, then this would result in an element with value +4. However, an element with value +4 may also result from 44 positive and 40 negative loops, or 6 positive and 2 negative loops, etc. Because the elements are a simple sum, more information is needed to interpret the result probabilistically; additionally, we need to know what the total number of feedback loops is.

Step 3: Find the absolute feedback matrix from the binary community matrix

To obtain the total number of feedback loops, we first create the binary community matrix. The binary community matrix, ${}^{\bullet}A$, has a `+1` for any interaction between species, regardless of its sign, and a `0` otherwise.

```
In [6]: # create binary community matrix
        Ab = matrix(QQ, Aq.nrows(), Aq.ncols())
        for position in Aq.nonzero_positions():
            Ab[position] = 1
        Ab
```

```
Out[6]: [1 1 0 0 0]
        [1 1 1 0 0]
        [0 1 1 1 1]
        [0 0 1 1 1]
        [0 0 0 1 1]
```

The total count of feedback loops is then obtained from the absolute feedback matrix, T. T is obtained from ${}^{\bullet}A$

$$T_{ji} = \mathrm{per}(\min {}^{\bullet}A_{ij}),$$

where 'per' is the matrix permanent, and where $\min {}^{\bullet}A_{ij}$ is the submatrix found by removing row i and column j of ${}^{\bullet}A$. The matrix permanent is calculated in a similar way to the determinant but with all terms in the sum having a positive sign.

```
In [7]: # absolute feedback matrix
        T = matrix(QQ,sz,sz)
        for row in range(sz):
            for col in range(sz):
```

```
            keeprows = range(sz)
            keeprows.remove(row)
            keepcols = range(sz)
            keepcols.remove(col)
            submatrix = Ab.matrix_from_rows_and_columns(keeprows, keepcols)
            T[col,row] = submatrix.permanent()

    T
```

```
Out[7]: [6 4 2 2 2]
        [4 4 2 2 2]
        [2 2 4 4 4]
        [1 1 2 3 5]
        [1 1 2 3 5]
```

Step 4: Use the adjoint of the qualitative community matrix and the absolute feedback matrix to find the weighted-predictions matrix

The weighted predictions matrix W is calculated by dividing the absolute value of each element of the adjoint by the corresponding element of T

$$W = \text{abs}(\text{adj}(-^{\circ}A))./T$$

where ./ indicates element-wise division. By convention, W_{ij} is set to 1 when $T_{ij} = 0$.

```
In [8]: # absolute value of adjugate matrix
        abs_adj = adj.apply_map(abs)

        # element-wise fraction of the adjugate to the
        # absolute feedback matrix
        fnc = lambda x: 0 if x == 0 else 1/x
        W = abs_adj.elementwise_product(T.apply_map(fnc))

        # Though for independent species,
        # we set the weighted predictions matrix value to 1
        fnc = lambda x: x == 0
        for position in (T.find(fnc, indices=True)).iterkeys():
            W[position] = 1
        W
```

```
Out[8]: [  1   1   1   1   0]
        [  1   1   1   1   0]
        [  1   1   0   0   0]
        [  1   1   0 1/3 1/5]
        [  1   1   0 1/3 1/5]
```

Interpret the weighted predictions matrix

From the analysis above, we have

$$\mathrm{adj}(-^\circ A) = \begin{bmatrix} +6 & -4 & +2 & +2 & 0 \\ -4 & +4 & -2 & -2 & 0 \\ -2 & +2 & 0 & 0 & 0 \\ +1 & -1 & 0 & +1 & -1 \\ +1 & -1 & 0 & +1 & +1 \end{bmatrix}, T = \begin{bmatrix} 6 & 4 & 2 & 2 & 2 \\ 4 & 4 & 2 & 2 & 2 \\ 2 & 2 & 4 & 4 & 4 \\ 1 & 1 & 2 & 3 & 5 \\ 1 & 1 & 2 & 3 & 5 \end{bmatrix}, W = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1/3 & 1/5 \\ 1 & 1 & 0 & 1/3 & 1/5 \end{bmatrix}$$

As explained in Dambacher et al. (2002), the elements in W, ranging between 0 and 1, indicate the determinacy of the response sign prediction obtained from $\mathrm{adj}(-^\circ A)$. A value of 1 indicates that all of the feedbacks between a row and column species are of the same sign. This means that the species response to a press perturbation, regardless of the specific interaction-strength values, is guaranteed to match the sign in $\mathrm{adj}(-^\circ A)$. Values less than one indicate indeterminacy: there is a combination of positive and negative feedback loops between the species in the system, and so the species response may be positive or negative, depending upon the specific interaction-strength values. A value of 0 indicates either that there are no feedbacks between the row and column species (see convention above), or that the number of positive and negative feedbacks are equal.

Dambacher et al. (2002) further interpreted the magnitude of $W_{ij} < 1$ values as indicative of the degree of indeterminacy in the species response. Based on Monte Carlo simulations, they recommended a threshold of $W_{ij} > 0.5$ as a general guideline for high ($\approx 95\%$ or more) sign determinacy (Dambacher et al. 2001). $W_{ij} > 0.5$ corresponds to $> 75\%$ of feedback loops having the same sign. Hosack et al. (2008) also used simulations, where interaction-strength values were sampled from various distributions, to obtain a relationship W_{ij} and sign-determinacy.

In the example in the paper, we are interested in predicting the effects of a negative press-perturbation of species 3 on species 3, 4, and 5. This corresponds to column 3, rows 3, 4, and 5 of the matrices above. In the weighted-predictions matrix, $W_{3,3} = W_{4,3} = W_{5,3} = 0$. Given that $T_{ij} \neq 0$ for the corresponding elements, this means that the response signs are maximally indeterminate: there are an equal number of positive and negative feedbacks between the perturbed species 3 and the other species.

```
In [9]: # the proportion of feedback loops that are positive
        fnc = lambda x: 0 if x == 0 else 1/x
        propn_pos = (1/2)*((adj+T).elementwise_product(T.apply_map(fnc)))
        propn_pos
```

```
Out[9]: [  1   0   1   1 1/2]
        [  0   1   0   0 1/2]
        [  0   1 1/2 1/2 1/2]
        [  1   0 1/2 2/3 2/5]
        [  1   0 1/2 2/3 3/5]
```

If the results of the weighted-predictions matrix analysis were interpreted probabilistically, then giving equal weighting to each feedback, the predictions are equivalent to coin flip between positive or negative response (c.f. Dambacher et al.'s (2002) 'old-field web' results).

C.1.2 Using Monte Carlo simulations and the Boolean approach

Define the species interaction network

We define the species interaction network:

- positive_edges_dict, a dictionary that specifies the positive interactions between species; and
- negative_edges_dict, a dictionary that specifies the negative interactions between species.

```
In [1]: # spp_list = ['s1','s2','s3','s4','s5']

        # key is recipient of a positive effect
        positive_edges_dict = {
        's5': ['s4'],
        's3': ['s2', 's4', 's5'],
        }

        # key is recipient of a negative effect
        negative_edges_dict = {
        's5': ['s5'],
        's4': ['s4', 's5', 's3'],
        's3': ['s3'],
        's2': ['s2', 's1', 's3'],
        's1': ['s1', 's2'],
        }
```

We encode the network structure as a networkx digraph. The function initialise_foodweb stores the signs of the interactions in the edge data dictionary.

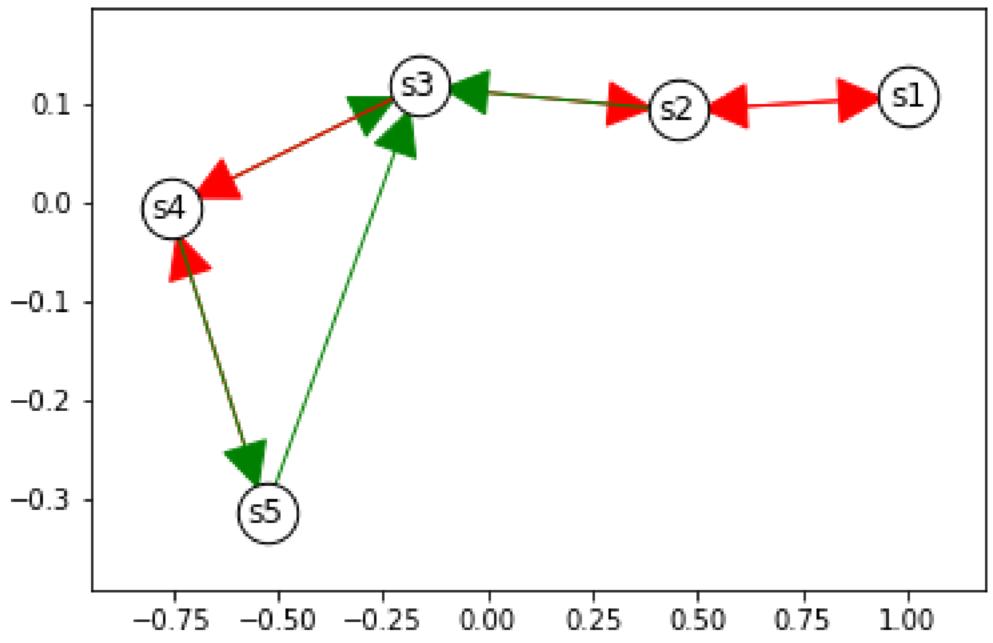
```
In [2]: from qualmod import initialise_foodweb

        web = initialise_foodweb(positive_edges_dict, negative_edges_dict)

        # print edges
        for edge in web.edges(data=True):
            print(edge)
```

```
('s5', 's5', {'color': 'red', 'sign': -1})
('s5', 's4', {'color': 'red', 'sign': -1})
```

```
('s5', 's3', {'color': 'green', 'sign': 1})
('s4', 's4', {'color': 'red', 'sign': -1})
('s4', 's5', {'color': 'green', 'sign': 1})
('s4', 's3', {'color': 'green', 'sign': 1})
('s3', 's4', {'color': 'red', 'sign': -1})
('s3', 's3', {'color': 'red', 'sign': -1})
('s3', 's2', {'color': 'red', 'sign': -1})
('s2', 's2', {'color': 'red', 'sign': -1})
('s2', 's1', {'color': 'red', 'sign': -1})
('s2', 's3', {'color': 'green', 'sign': 1})
('s1', 's2', {'color': 'red', 'sign': -1})
('s1', 's1', {'color': 'red', 'sign': -1})
```

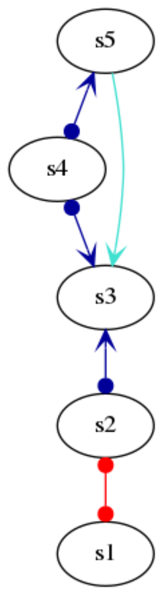
The package `networkx` provides a quick way to plot the interaction network.

```
In [3]: import networkx as nx
        import matplotlib.pyplot as plt

        pos=nx.spring_layout(web)
        nx.draw_networkx_labels(web, pos)
        nx.draw_networkx_nodes(web,
                               pos,
                               node_color = 'white',
                               edgecolors='black',
                               node_size=500)

        edge_colors = [ v[2]['color'] for v in web.edges(data=True) ]
        nx.draw_networkx_edges(web,
                               pos,
                               edge_color=edge_colors,
                               arrows=True,
                               arrowsize=40);
        plt.show()
```



The package `qualmod` also includes a function that uses `graphviz` to draw the interaction network. This is useful if we want to create a `.pdf` of the network to embed in a document, etc.

```
In [4]: from qualmod import draw_foodweb
        from IPython.display import IFrame
        from IPython.display import Image
        import os

        # use draw_foodweb to create a graphviz dot file
        # defining the interaction network
        draw_foodweb(web, f_name = 'web.dot')

        # call graphviz externally to create a pdf of the
        # interaction network
        os.system("dot -Tpdf web.dot > web.pdf")

        # display the pdf of the interaction network here in Jupyter
        # IFrame("web.pdf", width=500, height=600)

        # call graphviz to create a png, display in
        # markdown cell
        os.system("dot -Tpng web.dot > web.png")

Out[4]: 0
```



web.png

The interaction network can also be converted into a qualitative community matrix, `Mq` below, using the function `qualitative_community_matrix`.

```
In [5]: from qualmod import qualitative_community_matrix

        (Mq, s2idx) = qualitative_community_matrix(web)
        print(Mq)
```

```
[[-1. -1.  0.  0.  0.]
 [-1. -1. -1.  0.  0.]
 [ 0.  1. -1.  1.  1.]
 [ 0.  0. -1. -1. -1.]
 [ 0.  0.  0.  1. -1.]]
```

The function `qualitative_community_matrix` also returns `s2idx`, which is a handy bidirection dictionary that maps the name of the species to their index in the qualitative community matrix.

```
In [6]: print(s2idx)
        print(s2idx['s1'])
        print(s2idx.inv[0])
```

```
bidict({'s1': 0, 's2': 1, 's3': 2, 's4': 3, 's5': 4})
0
s1
```

Randomly sampling perturbation responses / parameter sweep

In the example in Fig. 2 of the paper, we are interested in the response of species `s3`, `s4`, and `s5` to a negative press-perturbation of species `s3`.

```
In [7]: # which species is controlled
        # (i.e. perturbed e.g. by pest control program)
        control_list = ['s3']

        # the species we're interested in the responses of
        monitored_list = ['s3','s4','s5']
```

In probabilistic Qualitative Modelling, the monitored species' response to the press-perturbation can be obtained using Monte Carlo simulation. The interaction-strength magnitudes in the community matrix are sampled from $a_{i,j} \sim \mathcal{U}(0, 1)$, and the positive and negative responses are counted: `countResponses` below.

In the Boolean approach, random sampling can also be used to do a parameter-value sweep of the model behaviour. The objective is to obtain a set, `observedResponseCombinations`, that lists every species-response combination that is possible, regardless of what the interaction-strength magnitudes are. Here note that, although we again sample from a uniform distribution, the choice is of distribution is somewhat arbitrary. So long as we obtain good-enough coverage of the parameter space that we obtain every possible response combination, then any sampling distribution can be used.

In this example, we apply only one plausibility criterion: the community matrix must be stable.

The method uses the sensitivity matrix approach to obtain species responses (`Sq`).

```
In [8]: import numpy as np

        # size of the system, can also use sz = web.order()
        sz = Mq.shape[0]

        # initialise collected responses with empty sets
        observedResponseCombinations = set()

        # initialise counts of positive responses with zeros
        countResponses = np.array([0]*len(control_list)*len(monitored_list))

        noSamples = 100
        for n in range(noSamples):

            # find a random community matrix that is stable
            maxEig = 1
            while maxEig > 0:

                M = np.multiply( np.random.random_sample((sz,sz)), Mq )
                maxEig = max(np.real(np.linalg.eigvals(M)))
            # find sensitivity matrix
            Sq = - np.linalg.inv(M)

            # Boolean approach:
            # ---

            #  find the signs of responses of each monitored species
            #  to perturbations in each control species
            response = tuple(['neg' if Sq[s2idx[ms],s2idx[cs]]<0 else 'pos'
                              for cs in control_list
                              for ms in monitored_list ])


            #  add response to our collection of observed responses
            observedResponseCombinations.add(response)

            # Probabilistic approach:
            # ---

            #  add positive responses to count
            posResponses = np.array([ 0 if Sq[s2idx[ms],s2idx[cs]]<0 else 1
                                      for cs in control_list
                                      for ms in monitored_list ])
            countResponses += posResponses
```

The two types of intermediate results are shown below. For the probabilistic approach, we have a count of positive responses in each species:

```
In [9]: countResponses
```

```
Out[9]: array([77, 23, 23])
```

For the Boolean approach, we have a list of species-response combinations:

```
In [10]: observedResponseCombinations
```

```
Out[10]: {('neg', 'pos', 'pos'), ('pos', 'neg', 'neg')}
```

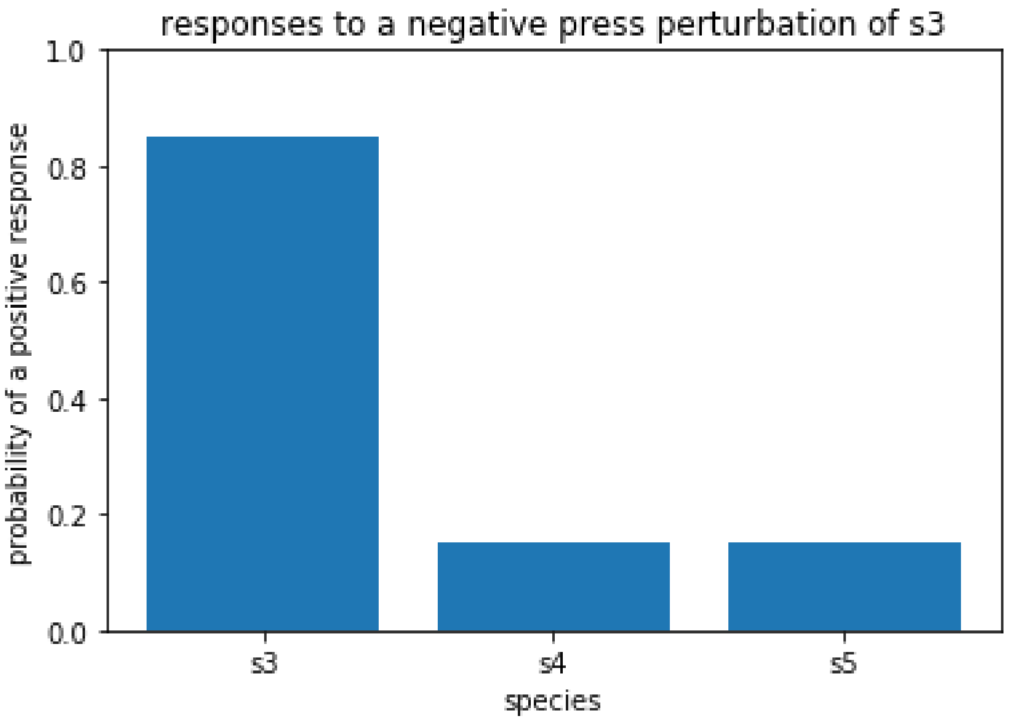
Probabilistic approach

In the probabilistic approach, the proportions of responses giving positive or negative species response are interpreted probabilistically.

In this example, `probabilityResponses` gives stronger support for a positive response in species `s3`, and gives stronger support for a negative response in species `s4` and `s5`.

```
In [11]: probabilityResponses = countResponses / noSamples

         plt.title('responses to a negative press perturbation of s3')
```

```
        plt.bar(range(len(monitored_list)),
                probabilityResponses,
                tick_label=monitored_list)
        plt.ylim((0,1))
        plt.ylabel('probability of a positive response')
        plt.xlabel('species')
        plt.show()
```



Boolean approach

In the Boolean approach, the species response-combinations that the model predicts are interpreted using Boolean algebra.

Two species-response combinations were observed during the parameter-value sweep (shown as a table below).

```
In [12]: # the responses we'll be counting proportions on
         desiredResponses = [ cs + '_' + ms
                             for cs in control_list
                             for ms in monitored_list]

         # I have hardcoded the answer in here in case, by chance,
         # you miss a response combination in the random sampling above.
         # For a real example, you will need to check that a large no.
```

```
        # of samples has been drawn since the last new response
        # combination was found.

        if len(observedResponseCombinations) < 2:
            observedResponseCombinations = {
                ('neg', 'pos', 'pos'),
                ('pos', 'neg', 'neg')
            }

        # print the species response combinations that were observed
        # during the parameter sweep above in a table
        header = ' | '.join(desiredResponses)
        print(header)
        print('-'*len(header))
        for responseCombination in observedResponseCombinations:
            print(' ' + '  |  '.join(responseCombination))
```

```
s3_s3 | s3_s4 | s3_s5
---------------------
 neg  |  pos  |  pos
 pos  |  neg  |  neg
```

In order to use Boolean algebra, we need to convert the species responses into Boolean values. The responses have a natural dichotimisation i.e. a positive or a negative species response. Arbitrarily, we assign a positive species response as a `true`, and a negative species response as a `false`.

```
In [13]: str4true = 'pos'; str4false = 'neg'
```

The simulations have returned the species-response combinations that are *possible* in the model, but the Boolean analysis is performed on the *impossible* combinations instead. If we assume that parameter sweep (above) was sufficient to obtain every possible combination, then the set of impossible combinations is simply the complement of the set of observed responses. In other words, the set of impossible combinations is the set of every response-combination that is combinatorially possible minus the set of responses that we observed.

There are a few different ways that we could obtain the complement, but one convenient way is to first observe that every response combination can be interpreted as a binary string, which can be interpreted as an integer. Then the complement corresponds to the list of integers that are missing from the set of integers that we observed.

First, we encode each observed response as a binary string, which is then converted into an integer:

```
In [14]: observedInts = [
             int( ''.join(['1' if i in str4true else '0'
                           for i in responseCombination]), 2 )
```

```
            for responseCombination in observedResponseCombinations
        ]
        observedInts
```

```
Out[14]: [3, 4]
```

Then, the unobserved species responses correspond to the integers that are missing from `observedInts`:

```
In [15]: # Start with set of all possible responses as integers ...
         unobservedInts = set(range(2**len(desiredResponses)))
         # ... and loop through the observed response combinations,
         # discarding those that were observed

         for i in observedInts:
             unobservedInts.discard(i)

         unobservedInts
```

```
Out[15]: {0, 1, 2, 5, 6, 7}
```

The function `getRespvarList2BoolvarList` converts the possible species-responses in the model into Boolean variables. It uses the PyEDA package.

```
In [16]: from findpcu import getRespvarList2BoolvarList

         # create our boolean variables and some useful dictionaries
         x, x2s, r2idx = getRespvarList2BoolvarList(desiredResponses,
                                                     str4true,
                                                     str4false)
```

`x` is a list of the Boolean variables that have been created. For example, `s3_s4` is the variable that describes the response of species s4 to a negative perturbation of s3. `s3_s4` has the value `True` if s4 responds positively and `False` if s4 responds negatively.

```
In [17]: x
```

```
Out[17]: [s3_s3, s3_s4, s3_s5]
```

`x2s` and `r2idx` are a bidirectional dictionaries that convert between a variable values and a corresponding string or index, respectively.

The next step is to use the function `intList2boolexpr` to turn the list of integers `unobservedInts`, corresponding to unobserved species responses, into a Boolean expression.

```
In [18]: from findpcu import intList2boolexpr

         # the boolean expression for our unobserved responses
         unobservedBoolexpr = intList2boolexpr(unobservedInts, x)
```

The Boolean expression is returned in disjunctive normal form, as a sum of products. Here the notation ~ is used to represent Not. For example, `~s3_s4` means 'not a positive response in `s4` to a negative press-perturbation in `s3`', or in other words, 'a negative response in `s4`'.

```
In [19]: print(unobservedBoolexpr)
```

```
Or(
        And(~s3_s3, ~s3_s4, ~s3_s5),
        And(~s3_s3, ~s3_s4, s3_s5),
        And(~s3_s3, s3_s4, ~s3_s5),
        And(s3_s3, ~s3_s4, s3_s5),
        And(s3_s3, s3_s4, ~s3_s5),
        And(s3_s3, s3_s4, s3_s5)
)
```

The list of unobserved species responses can also be easily converted into a truth table using the function `expr2truthtable` from PyEDA. This is the same table as given in Fig. 2a in the paper.

```
In [20]: from pyeda.inter import expr2truthtable
         expr2truthtable(unobservedBoolexpr)
```

```
Out[20]: s3_s5 s3_s4 s3_s3
             0     0     0 : 1
             0     0     1 : 0
             0     1     0 : 1
             0     1     1 : 1
             1     0     0 : 1
             1     0     1 : 1
             1     1     0 : 0
             1     1     1 : 1
```

In order to summarise the result, we perform a Boolean minimisation using the espresso algorithm.

```
In [21]: from pyeda.inter import espresso_exprs

         # Use espresso to minimise the unobservedBoolexpr
         boolExprMin, = espresso_exprs(unobservedBoolexpr)
```

When we print out the resulting minimised Boolean expression, we see that it is shorter and simpler than the original (compare to `unobservedBoolexpr` above).

```
In [22]: boolExprMin
```

```
Out[22]: Or(And(~s3_s4, s3_s5), And(s3_s3, s3_s4), And(~s3_s3, ~s3_s5))
```

Note however that `boolExprMin` is equivalent to the original expression. We can verify that by printing out its (full) truth table (using `expr2truthtable` again) and comparing it to the one above.

```
In [23]: expr2truthtable(boolExprMin)
```

```
Out[23]: s3_s5 s3_s4 s3_s3
             0     0     0 : 1
             0     0     1 : 0
             0     1     0 : 1
             0     1     1 : 1
             1     0     0 : 1
             1     0     1 : 1
             1     1     0 : 0
             1     1     1 : 1
```

Each of the `And` terms in the minimised Boolean expression (`boolExprMin`) is called a PCU (short for the French ”projection canonique ultime” meaning ”ultimate canonical projection”), and Theuns (1994) discusses some of the theory behind them.

The function `boolexpr2RespvalList` converts each of the PCUs into a string, which can be useful if we wish to store the results in a file. This function also appends the strings (in our case `pos` and `neg`) that we designated above as `True` and `False`, to make the output quicker to read.

```
In [24]: from findpcu import boolexpr2RespvalList

         # returns the PCUs of the boolean expression as a list of strings
         PCUList = boolexpr2RespvalList(boolExprMin, x2s)

         PCUList
```

```
Out[24]:
[
    ['poss3_s3', 'poss3_s4'],
    ['negs3_s5', 'negs3_s3'],
    ['poss3_s5', 'negs3_s4']
]
```

Each PCU can be used to derive a set of logical implications, and these implications can be chained together to create a logical implication network.

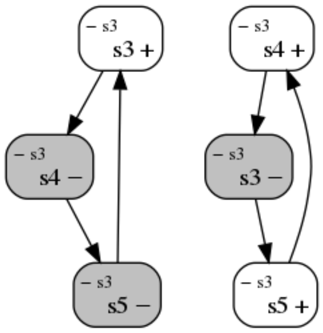
All that is needed to ensure that the logical implication network describes the full behaviour of the model is to ensure that at least one implication is derived from each PCU. However, for small networks, it may be preferable to include more than one implication. The function `draw_implication_network` draws the network in such a way that every implication with 1 antecedent is included.

```
In [25]: from findpcu import draw_implication_network

         draw_implication_network(PCUList)
```

```
# display the pdf of the interaction network here in Jupyter
# IFrame("implication_network.pdf", width=500, height=500)

# call graphviz to create a png, display in markdown cell
os.system("dot -Tpng implication_network.dot > implication_network.png")
```

```
implication_network.pdf has been created
```

```
Out[25]: 0
```



implication_network.png

We can see that each PCU is represented twice in the implication network above. For example, `['poss3_s3', 'poss3_s4']` translates to saying that the following combination of species responses is impossible in the model: positive s3 and positive s4. Therefore a positive response in s3 implies a negative response in s4 (seen in left-hand subnetwork), and a positive response in s4 implies a negative response in s3 (seen in right-hand subnetwork).

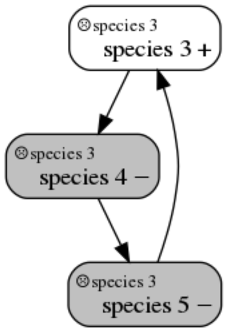
Another function is available, `draw_implication_network2`, to draw the implication network with only one implication per PCU. We can specify which species responses we wish to see as the antecedents. The example below also shows some of its other options.

```
In [26]: from findpcu import draw_implication_network2

         # make a positive response of s3 always an antecedent
         alwaysAnteList = ['poss3_s3', 'negs3_s5', 'negs3_s4']

         niceNames = { # names of each species
             's3': 'species 3',
             's4': 'species 4',
             's5': 'species 5',
         }

         fName = 'implication_network_2' # name of the .dot file and .pdf file

         controlSymbol = '&#9785;' # html code for a sad face
```

```
draw_implication_network2(PCUList,
                          alwaysAnteList,
                          fName,
                          niceNames,
                          controlSymbol)

os.system("dot -Tpng implication_network_2.dot > implication_network_2.png")
```

```
implication_network_2.pdf has been created
```

```
Out[26]: 0
```



```
implication_network_2.png
```

C.2 Implementing the Boolean approach on the Macquarie Island case study

The purpose of this tutorial is to explain how a Boolean analysis of the Macquarie Island case study was performed.

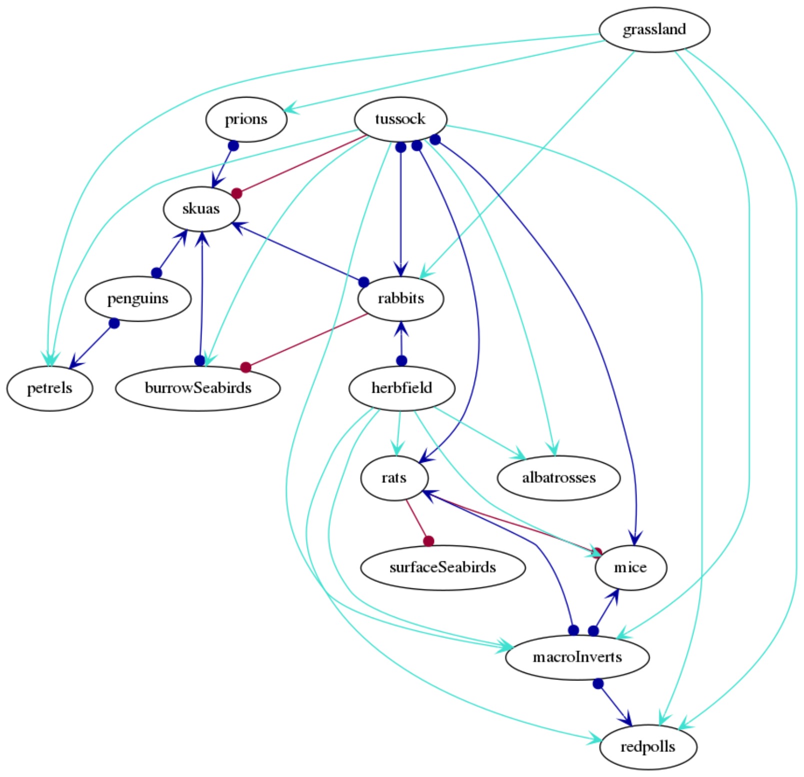
This Jupyter Notebook is archived as: Tutorial-2-Macquarie_Island_case_study.ipynb

Notebook

We define the Macquarie Island interaction network by the species present, the interactions between them, and the signs of these interactions. The network structure is encoded as a `networkx` digraph. The function `initialise_foodweb` stores the signs of the interactions in the edge data dictionary. The network can be drawn using `draw_foodweb`.

```
In [90]: from qualmod import initialise_foodweb, draw_foodweb
         from qualmod import qualitative_community_matrix, get_conditions_lists
         import numpy as np
         import time

         from IPython.display import Image
         import os

         sppList = [
             'albatrosses',
             'prions',
             'burrowSeabirds',
             'petrels',
             'herbfield',
             'macroInverts',
             'mice',
             'penguins',
             'rabbits',
             'rats',
             'redpolls',
             'skuas',
             'surfaceSeabirds',
             'tussock',
         ]

         # key is recipient of positive effect
         positive_edges_dict = {
             'prions':           ['grassland'],
             'skuas':            ['prions','burrowSeabirds','rabbits','penguins'],
             'petrels':          ['penguins','tussock','grassland'],
             'mice':             ['herbfield','macroInverts','tussock'],
             'rats':             ['macroInverts','herbfield','tussock'],
             'burrowSeabirds':   ['tussock'],
             'rabbits':          ['tussock','herbfield','grassland'],
             'macroInverts':     ['herbfield','grassland','tussock'],
             'albatrosses':      ['tussock','herbfield'],
             'redpolls':         ['macroInverts','tussock','herbfield','grassland'],
         }
         negative_edges_dict = {
             'prions':           ['prions','skuas'],
             'skuas':            ['skuas','tussock'],
             'penguins':         ['penguins','skuas','petrels'],
             'petrels':          ['petrels'],
             'mice':             ['mice','rats'],
             'rats':             ['rats'],
             'burrowSeabirds':   ['burrowSeabirds','skuas','rabbits'],
             'rabbits':          ['rabbits','skuas'],
             'surfaceSeabirds':  ['surfaceSeabirds','rats'],
             'macroInverts':     ['macroInverts','rats','mice','redpolls'],
             'tussock':          ['tussock','mice','rats','rabbits'],
             'albatrosses':      ['albatrosses'],
             'herbfield':        ['herbfield','rabbits'],
             'grassland':        ['grassland'],
             'redpolls':         ['redpolls'],
         }

         web = initialise_foodweb(positive_edges_dict, negative_edges_dict)

         draw_foodweb(web, f_name = 'macq1.dot')
         # call graphviz to create a png, display in markdown cell
         os.system("dot -Tpng macq1.dot > macq1.png")
```

```
Out[90]: 0
```



Macquarie Island interaction network.

We encode the validation criteria (with respect to a positive perturbation of the rabbits).

```
In [74]: control_list = ['rabbits']

         # Reponse to increase in rabbits
         validation = {
             'rabbits': +1,
             'tussock': -1,
         }
```

We convert the interaction network into a qualitative community matrix, `Mq` below, using the function `qualitative_community_matrix`.

```
In [78]: sz = web.order()
         (Mq, s2idx) = qualitative_community_matrix(web)
         print(Mq)
```

```
[[-1.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0. -1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.  1.]
 [ 0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  1. -1. -1.  0.  0.  0.  0. -1. -1.  0.  0.  1.]
 [ 0.  0.  0.  1.  1. -1.  0.  0.  0.  0. -1.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. -1. -1.  0.  0.  0.  0. -1.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  1. -1.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0. -1.  0.  0.]
 [ 0.  0.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.  1.]
 [ 0.  0.  0.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0.  1.]
 [ 0.  0.  1.  1.  1.  0.  0.  0.  0.  0.  0. -1.  0.  0.  1.]
 [ 0.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0. -1.  0. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0. -1.  0.]
 [ 0.  0.  0.  0.  0. -1.  0.  0.  0. -1. -1.  0.  0.  0. -1.]]
```

The `s2idx` returned is a bidirection dictionary that maps the name of the species to their index in the qualitative community matrix

```
In [79]: s2idx.inv[9]

Out[79]: 'rabbits'
```

The validation conditions with respect to the indices of the qualitative matrix can be obtained using the function `get_conditions_list`.

```
In [77]: # validation condition
         condn_idxs, condn = get_conditions_lists(s2idx, validation)

         print(list(zip(condn_idxs, condn)))

[(9, 1), (14, -1)]
```

To save time in this tutorial, I have hard-coded in the number of species-response combinations to be found, which is 36. If you do not know what the number is beforehand, you would use a line similar to the one commented-out above the start of the `while` loop.

The `while` loop collects species response combinations using a parameter-value sweep of the model.

```
In [91]: # Initialise collected responses with empty sets
         # Key is species perturbed and set will contain tuples of responses
         collectedResponses = {spp: set() for spp in control_list}

         t = 1
         t_last_updated = 0
         start_time = time.time()

         # use something like this if you don't
         # know the answer beforehand ~ 10 minutes
         # while t < 1e6:

         # I know there are 36 responses to find
         while len(collectedResponses['rabbits']) < 36:
```

```
    # Find valid stable community matrix
    valid = False

    while not valid:

        # Find a random community matrix that is stable
        maxEig = 1

        while maxEig > 0:
            M = np.multiply(np.random.random_sample((sz,sz)), Mq)
            maxEig = max(np.real(np.linalg.eigvals(M)))

        # Now have a stable matrix

        # Find sensitivity matrix
        Sq = - np.linalg.inv(M)

        # Check validation criteria
        valid = all(np.sign(Sq[condn_idxs,s2idx['rabbits']]) == condn)

    # Now have a valid stable community matrix

    for ps in control_list:

        # NOTE: looking at increase in rabbits not decrease here
        response = tuple('neg' if Sq[s2idx[ms],s2idx[ps]]<0 else 'pos'
                         if Sq[s2idx[ms],s2idx[ps]]>0 else 'zer'
                         for ms in sppList)

        if response not in collectedResponses[ps]:

            # fs[ps].write(','.join(response) + '\n')
            collectedResponses[ps].add(response)
            t_last_updated = t

    t += 1

end_time = time.time()
time_elapsed = end_time - start_time
print('time elapsed = ' + str(time_elapsed) + ' seconds')
print('number of matrices sampled = ' + str(t-1))
print('last new combination found at t = ' + str(t_last_updated))
```

```
time elapsed = 0.5220227241516113 seconds
number of matrices sampled = 889
last new combination found at t = 889
```

The possible species-response combinations are now stored in `collectedResponses['rabbits']`. If more than one control-species is involved, then its name would be the key for the `collectedResponses` dictionary.

```
In [95]: print( ' '.join([ ss[:3] for ss in sppList]))
         for response in collectedResponses['rabbits']:
             print(' '.join(response))
```

```
alb pri bur pet her mac mic pen rab rat red sku sur tus
neg pos neg neg neg pos neg pos pos pos neg neg neg neg
neg pos neg neg neg neg neg pos pos neg neg neg pos neg
neg neg neg neg neg neg neg neg pos neg neg pos pos neg
neg neg neg neg neg neg pos pos pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos neg pos neg pos neg
neg pos neg neg neg pos neg pos pos pos pos neg neg neg
neg pos neg pos neg pos neg pos pos neg pos neg pos neg
neg pos neg pos neg pos pos pos pos neg neg neg pos neg
neg pos neg pos neg pos pos pos pos pos neg neg neg neg
neg pos neg pos neg neg pos pos pos neg neg neg pos neg
neg pos neg pos neg pos neg pos pos pos neg neg neg neg
neg pos neg pos neg neg neg pos pos neg neg neg pos neg
neg neg neg neg neg neg pos neg pos neg neg pos pos neg
neg pos neg pos neg pos neg pos pos neg neg neg pos neg
neg pos neg pos neg pos pos pos pos neg pos neg pos neg
neg neg neg neg neg neg neg pos pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos pos neg neg neg neg
neg pos neg neg neg neg pos pos pos neg neg neg pos neg
neg neg neg neg neg pos pos pos pos pos neg pos neg neg
neg neg neg neg neg pos neg neg pos pos neg pos neg neg
neg neg neg neg neg pos neg pos pos pos pos pos neg neg
neg pos neg pos neg pos neg pos pos pos pos neg neg neg
neg neg neg neg neg pos neg pos pos pos neg pos neg neg
neg neg neg neg neg pos neg neg pos pos pos pos neg neg
neg neg neg neg neg pos pos neg pos pos neg pos neg neg
neg neg neg neg neg pos pos pos pos neg neg pos pos neg
neg neg neg neg neg pos neg neg pos neg neg pos pos neg
neg pos neg neg neg pos pos pos pos neg neg neg pos neg
neg pos neg neg neg pos neg pos pos neg pos neg pos neg
neg neg neg neg neg pos pos neg pos neg neg pos pos neg
neg neg neg neg neg pos neg pos pos neg neg pos pos neg
neg neg neg neg neg pos pos pos pos neg pos pos pos neg
neg neg neg neg neg pos neg neg pos neg pos pos pos neg
neg neg neg neg neg pos neg pos pos neg pos pos pos neg
neg neg neg neg neg pos pos neg pos neg pos pos pos neg
neg pos neg neg neg pos neg pos pos neg neg neg pos neg
```

At this point in the process, it is a good idea to write the model responses. Here we write them to a csv file.

```
In [96]: # write output files

         fs = {ps: open('uniques_web1_' + ps + '.csv', 'w')
               for ps in control_list}

         for ps in control_list:

             header = [ps + '_' + ms for ms in sppList]
             fs[ps].write(','.join(header) + '\n')

             for response in collectedResponses[ps]:

                 fs[ps].write(','.join(response) + '\n')

             fs[ps].close()

         print(header)
```

```
['rabbits_albatrosses', 'rabbits_prions', 'rabbits_burrowSeabirds',
        'rabbits_petrels', 'rabbits_herbfield', 'rabbits_macroInverts',
        'rabbits_mice', 'rabbits_penguins', 'rabbits_rabbits', 'rabbits_rats',
        'rabbits_redpolls', 'rabbits_skuas', 'rabbits_surfaceSeabirds',
        'rabbits_tussock']
```

From here onwards, we will be performing the Boolean analysis on the species responses that were found in the model.

We read in the responses from the csv file that was written previously.

```
In [92]: import csv
         from pyeda.inter import espresso_exprs
         from findpcu import getUnobservedInts, getRespvarList2BoolvarList
         from findpcu import intList2boolexpr, boolexpr2RespvalList

         fInName = ('uniques_web1_rabbits.csv')
         str4true = 'pos'
         str4false = 'neg'
         fIn = open(fInName)
         csv_f = csv.reader(fIn)
         allResponses = next(csv_f) # get the header
         fIn.close()

         allResponses
```

```
Out[92]: ['rabbits_albatrosses',
          'rabbits_prions',
          'rabbits_burrowSeabirds',
          'rabbits_petrels',
          'rabbits_herbfield',
          'rabbits_macroInverts',
          'rabbits_mice',
          'rabbits_penguins',
          'rabbits_rabbits',
          'rabbits_rats',
          'rabbits_redpolls',
          'rabbits_skuas',
          'rabbits_surfaceSeabirds',
          'rabbits_tussock']
```

We already know that the response of tussock to rabbits will be negative, so we write a list of desired responses, in the same order as `allResponses`, that omits tussock. The `desiredResponses` list is converted into a Boolean mask, and passed to the function `unobservedInts`, which uses the list of observed responses and desired responses to return a list of unobserved responses as a list of integers.

```
In [97]: # skip tussock, because its response was a validation criterion
         desiredResponses = [
             'rabbits_albatrosses',
             'rabbits_prions',
             'rabbits_burrowSeabirds',
             'rabbits_petrels',
             'rabbits_herbfield',
             'rabbits_macroInverts',
             'rabbits_mice',
             'rabbits_penguins',
             'rabbits_rats',
             'rabbits_redpolls',
             'rabbits_skuas',
             'rabbits_surfaceSeabirds']

         boolLen = len(desiredResponses)
```

```
        # A mask to take out only those entries in our desiredResponses
        desiredResponsesMask = [True if aR in desiredResponses
                                else False for aR in allResponses]

        unobservedInts = getUnobservedInts(fInName,
                                           desiredResponsesMask,
                                           boolLen, str4true)
```

```
In [85]: len(unobservedInts)
```

```
Out[85]: 4060
```

The function `getRespvarList2BoolvarList` is used to convert the possible species-responses in the model into Boolean variables. It uses the PyEDA package (https://pyeda.readthedocs.io/en/latest/).

The function `intList2boolexpr` turns the list of integers `unobservedInts`, corresponding to unobserved species responses, into a Boolean expression.

```
In [98]: # Create our boolean variables and some useful dictionaries
         x, x2s, r2idx = getRespvarList2BoolvarList(desiredResponses,
                                                    str4true, str4false)

         # Turn each integer representing unobserved into a
         # into boolean expression
         unobservedBoolexpr = intList2boolexpr(unobservedInts, x)
```

The function `espresso_exprs`, imported from PyEDA, is used to perform the Boolean minimisation.

```
In [87]: start_time = time.time()

         boolExprMin, = espresso_exprs(unobservedBoolexpr)

         end_time = time.time()
         time_elapsed = end_time - start_time # ~ 1 second
         print(str(time_elapsed))

1.2536895275115967
```

This results in a Boolean expression that summarises the species responses that were not observed in the parameter-sweep of the model, and are assumed to be impossible.

```
In [88]: boolExprMin
```

```
Out[88]: Or(rabbits_albatrosses, rabbits_burrowSeabirds, rabbits_herbfield,
    And(rabbits_mice, rabbits_redpolls, ~rabbits_surfaceSeabirds),
    And(~rabbits_rats, ~rabbits_surfaceSeabirds),
    And(rabbits_petrels, rabbits_skuas),
    And(~rabbits_macroInverts, rabbits_redpolls),
    And(~rabbits_prions, ~rabbits_skuas),
    And(~rabbits_macroInverts, ~rabbits_surfaceSeabirds),
    And(rabbits_prions, rabbits_skuas),
    And(rabbits_rats, rabbits_surfaceSeabirds),
    And(~rabbits_penguins, ~rabbits_skuas))
```

The function `boolexpr2RespvalList` converts each of the PCUs into a list of strings, which can be useful if we wish to store the results in a file, and is easier to read.

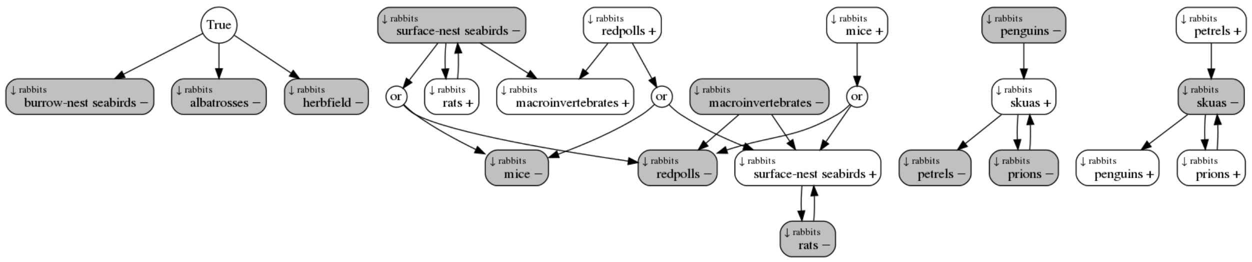
```
In [89]: PCUList = boolexpr2RespvalList(boolExprMin, x2s)
         PCUList

Out[89]: [['posrabbits_burrowSeabirds'],
          ['posrabbits_albatrosses'],
          ['posrabbits_herbfield'],
          ['negrabbits_surfaceSeabirds', 'negrabbits_rats'],
          ['negrabbits_penguins', 'negrabbits_skuas'],
          ['negrabbits_macroInverts', 'posrabbits_redpolls'],
          ['posrabbits_skuas', 'posrabbits_petrels'],
          ['negrabbits_skuas', 'negrabbits_prions'],
          ['posrabbits_prions', 'posrabbits_skuas'],
          ['posrabbits_rats', 'posrabbits_surfaceSeabirds'],
          ['negrabbits_macroInverts', 'negrabbits_surfaceSeabirds'],
          ['posrabbits_redpolls', 'negrabbits_surfaceSeabirds', 'posrabbits_mice']]
```

The `PCUList` form is also used as an input to the `draw_implication_network` function, which draws all logical implications resulting from the Boolean minimisation in their single-antecedent form.

```
In [100]: import os
          from findpcu import draw_implication_network

          niceNames = {
                  'rabbits': 'rabbits',
                  'petrels': 'petrels',
                  'mice': 'mice',
                  'burrowSeabirds': 'burrow-nest seabirds',
                  'macroInverts': 'macroinvertebrates',
                  'herbfield': 'herbfield',
                  'redpolls': 'redpolls',
                  'skuas': 'skuas',
                  'rats': 'rats',
                  'surfaceSeabirds': 'surface-nest seabirds',
                  'penguins': 'penguins',
                  'prions': 'prions',
                  'albatrosses': 'albatrosses',
                  'tussock': 'tussock'
                 }

         controlSymbol = '&darr; '

         draw_implication_network(PCUList,
                                   'uniques_web1_rabbits_pcus',
                                  niceNames, controlSymbol)

         # call graphviz to create a png, display in markdown cell
         cmd = "dot -Tpng uniques_web1_rabbits_pcus.dot > uniques_web1_rabbits_pcus.png
         os.system( cmd )
```

```
uniques_web1_rabbits_pcus.pdf has been created
```

```
Out[100]: 0
```



Implication network.

C.3 Dealing with complicated implication networks

Large implication networks can be difficult to interpret, even when the individual subnetworks are small. In such cases, it may be necessary to separate the subnetworks and rearrange them for a useful interpretation. In this tutorial, we give examples of how decisions about the presentation of the network may be made, and how that may influence the form of the final network presented to conservation decision-makers. We use the Macquarie Island model again as the example, but this time consider the outcomes of rat control.

This Jupyter Notebook is archived as:
Tutorial-3-Dealing_with_complicated_implication_networks.ipynb

Notebook

In this example, we'll use the list of PCU's below (see Tutorial 2 for how to obtain PCUs) that were obtained from simulating rat control on Macquarie Island.

```
In [1]: PCUList = [
          ['posrats_surfaceSeabirds'],
          ['negrats_rabbits', 'negrats_herbfield'],
          ['negrats_prions', 'negrats_skuas'],
          ['posrats_herbfield', 'posrats_rabbits'],
          ['posrats_prions', 'posrats_skuas'],
          ['posrats_tussock', 'posrats_mice',
           'posrats_rabbits'],
          ['negrats_rabbits', 'negrats_burrowSeabirds',
           'posrats_tussock'],
          ['negrats_penguins', 'negrats_skuas',
           'negrats_tussock'],
          ['negrats_rabbits', 'posrats_tussock',
           'negrats_skuas'],
          ['posrats_skuas', 'posrats_petrels',
           'negrats_tussock'],
          ['posrats_redpolls', 'posrats_mice',
           'posrats_rabbits'],
          ['posrats_tussock', 'negrats_petrels',
           'negrats_skuas'],
          ['negrats_tussock', 'posrats_burrowSeabirds',
           'posrats_rabbits'],
          ['negrats_macroInverts', 'posrats_mice',
           'posrats_rabbits'],
          ['posrats_albatrosses', 'negrats_tussock',
           'posrats_rabbits'],
          ['posrats_skuas', 'negrats_tussock',
           'posrats_rabbits'],
          ['posrats_tussock', 'posrats_skuas',
           'posrats_penguins'],
          ['negrats_rabbits', 'posrats_tussock',
           'negrats_albatrosses'],
          ['negrats_rabbits', 'posrats_tussock',
           'negrats_redpolls', 'posrats_macroInverts'],
          ['negrats_macroInverts', 'posrats_redpolls',
           'negrats_tussock', 'posrats_rabbits'],
          ['posrats_petrels', 'posrats_mice', 'posrats_macroInverts',
           'posrats_redpolls', 'negrats_albatrosses'],
          ['posrats_petrels', 'posrats_mice',
           'posrats_burrowSeabirds', 'posrats_redpolls',
           'negrats_albatrosses'],
          ['posrats_mice', 'posrats_burrowSeabirds',
           'posrats_macroInverts', 'posrats_redpolls',
           'negrats_albatrosses', 'negrats_skuas']
         ]
```

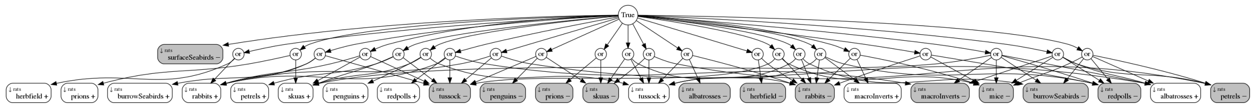
Our objective is to draw an implication network that can be interpreted for decision-makers planning a rat control programme.

In the first attempt, the implication network that results is very complicated and difficult to interpret.

```
In [3]: from findpcu import draw_implication_network, draw_implication_network2
        import os

        draw_implication_network2(PCUList,
                                  [],
                                  'attempt_1',
                                  niceNames = None,
                                  controlSymbol = 'downarrow')

        # call graphviz to create a png, display in markdown cell
        os.system("dot -Tpng attempt_1.dot > attempt_1.png")
```

```
attempt_1.pdf has been created
```

```
Out[3]: 0
```



```
PCUList
```

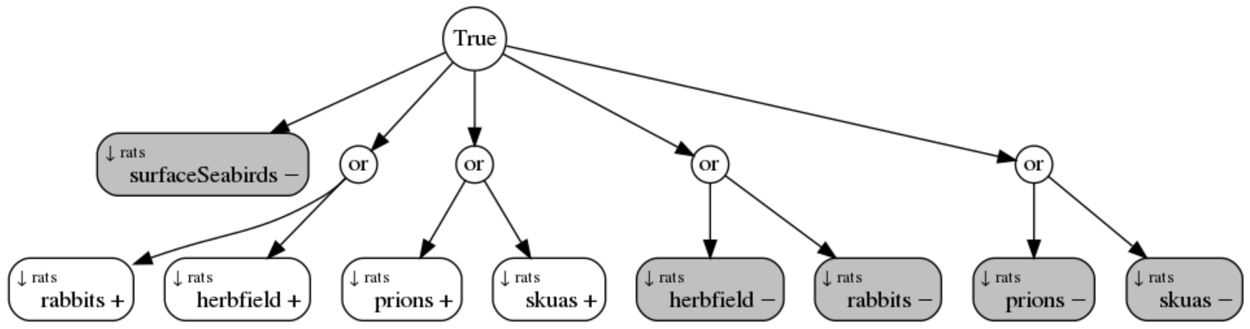
To simplify a network, a good first step is to split the PCUs up by length and present them in separate networks. Provided that every PCU is represented in the networks, then no information will be lost.

It is also a good idea to put highly-connected responses, and other pests, at the top as antecedents. We can spot highly-connected nodes by the number of arrows that are feeding into them, and placing other pest species at the top may be useful to decision-makers who are considering either flow-on effects on other pests, or a multi-species control programme.

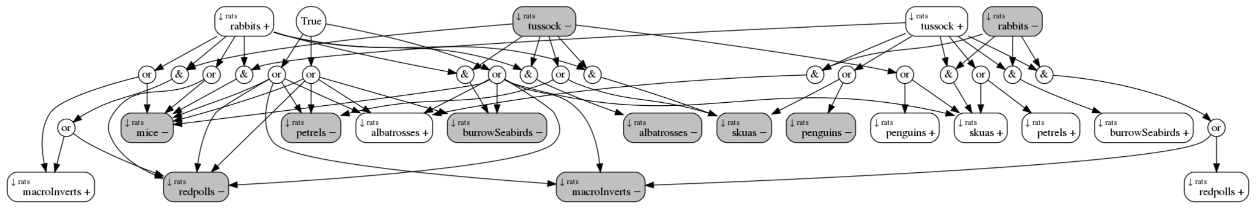
```
In [11]: PCUList1 = [ PCU for PCU in PCUList if len(PCU) <= 2 ]
         PCUList2 = [ PCU for PCU in PCUList if len(PCU) > 2 ]

         draw_implication_network2(PCUList1,
             [],
             'PCUList1_1', niceNames = None, controlSymbol = 'downarrow')
         draw_implication_network2(PCUList2,
             ['negrats_rabbits', 'posrats_rabbits', 'posrats_tussock', 'negrats_tussock'
             'PCUList2_1', niceNames = None, controlSymbol = 'downarrow')

         os.system("dot -Tpng PCUList1_1.dot > PCUList1_1.png")
         os.system("dot -Tpng PCUList2_1.dot > PCUList2_1.png")
```

```
PCUList1_1.pdf has been created
PCUList2_1.pdf has been created
```

```
Out[11]: 0
```



PCUList1_1



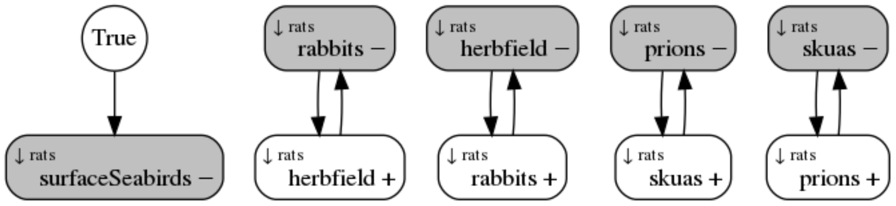
PCUList2_1

In the top network (PCUList1), we recognise the same bidirectional implication relationship that appeared previously, in the network representing rabbit control. Rabbits and herbfield are also in a bidirectional relationship. For short implications, the function draw_implication_network can be more useful. It will repeat the information contained in some of the PCUs (e.g. showing both bidirectional outcomes), but when the implications are short, this doesn't increase the complexity of the network much compared to the increase in clarity of the implications' meaning.

```
In [5]: draw_implication_network(PCUList1,
                                 'PCUList1_2',
                                 niceNames = None,
                                 controlSymbol = 'downarrow')
        os.system("dot -Tpng PCUList1_2.dot > PCUList1_2.png")
```

```
PCUList1_2.pdf has been created
```

```
Out[5]: 0
```



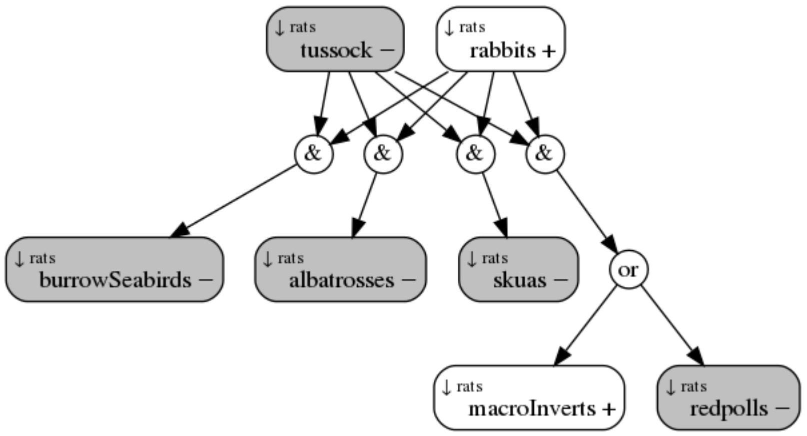
PCUList1_2

In the second network (`PCUList2`), we see that a symmetry in the effect of the combined response in rabbits and tussock. We can split those out into two separate networks and place the remainder of the PCUs in `PCUList5` for further analysis.

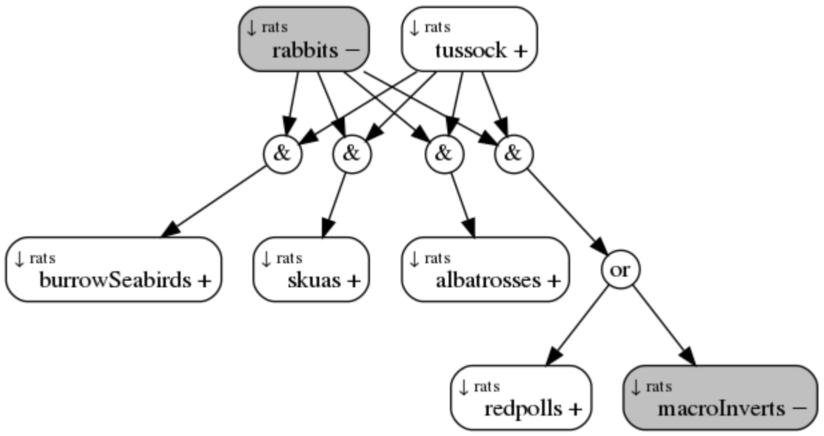
```
In [6]: PCUList3 = list() # rat and tussock relationships
        PCUList4 = list()
        PCUList5 = list() # other
        for PCU in PCUList2:

            if 'posrats_rabbits' in PCU and 'negrats_tussock' in PCU:
                PCUList3.append(PCU)
            elif 'negrats_rabbits' in PCU and 'posrats_tussock' in PCU:
                PCUList4.append(PCU)
            else:
                PCUList5.append(PCU)

        draw_implication_network2(PCUList3,
                    ['posrats_rabbits', 'negrats_tussock'],
                    'PCUList3_1', niceNames = None, controlSymbol = 'downarrow')
        draw_implication_network2(PCUList4,
                    ['negrats_rabbits', 'posrats_tussock'],
                    'PCUList4_1', niceNames = None, controlSymbol = 'downarrow')
        os.system("dot -Tpng PCUList3_1.dot > PCUList3_1.png")
        os.system("dot -Tpng PCUList4_1.dot > PCUList4_1.png")
```

```
PCUList3_1.pdf has been created
PCUList4_1.pdf has been created
```

```
Out[6]: 0
```



PCUList3_1



PCUList4_1

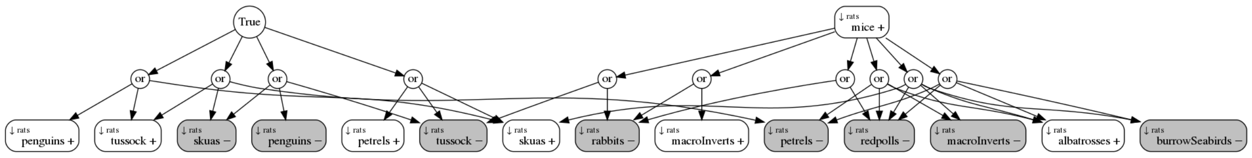
The symmetry between the two scenarios above may be useful for understanding the whole-system behaviour. For example, the response of burrow-nesting seabirds and albatrosses echoes what was found for rabbit control.

We now take a closer look at the PCUs that remain.

```
In [7]: draw_implication_network2(PCUList5,
                    ['posrats_mice'],
                    'PCUList5_1', niceNames = None, controlSymbol = 'downarrow')
        os.system("dot -Tpng PCUList5_1.dot > PCUList5_1.png")
```

```
PCUList5_1.pdf has been created
```

```
Out[7]: 0
```



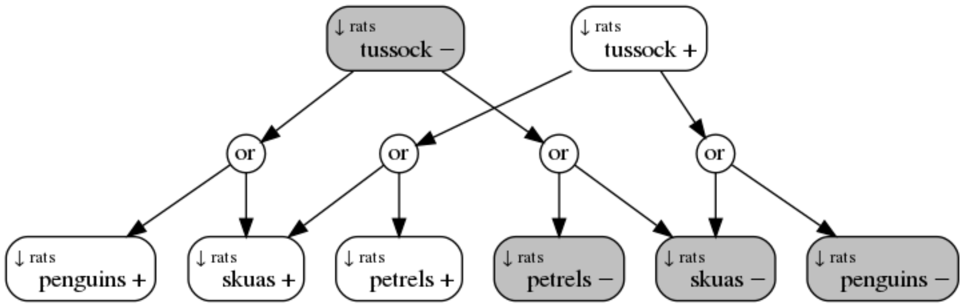
```
PCUList5_1
```

There is an obvious split between relationships involving mice and not. The relationships not involving mice have a symmetry, so we should choose an arrangement that emphasises that. Below, we choose to place the effect of rat control on tussock as the antecedent.

```
In [8]: PCUList6 = list() # for rules without mice
        PCUList7 = list() # for rules with mice

        for PCU in PCUList5:
            if 'posrats_mice' in PCU:
                PCUList7.append(PCU)
            else:
                PCUList6.append(PCU)

        draw_implication_network2(PCUList6,
                    ['posrats_tussock', 'negrats_tussock'],
                    'PCUList6_1', niceNames = None, controlSymbol = 'downarrow')
        os.system("dot -Tpng PCUList6_1.dot > PCUList6_1.png")
```

```
PCUList6_1.pdf has been created
```

```
Out[8]: 0
```



```
PCUList6_1
```

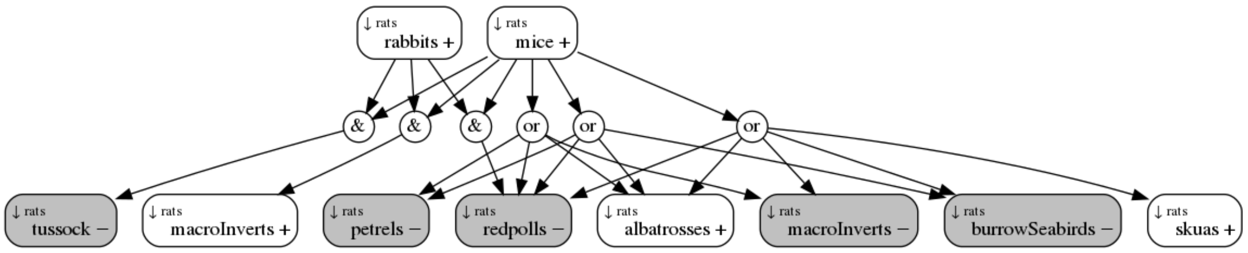
Now we consider the right-hand side, the relationships involving mice.

The importance of tussock suggests that the interaction between rabbits, mice, and tussock may be interesting in its own right. The simultaneous effect on rabbits is also involved in outcomes for macroinvertebrates and redpolls, and rabbits are another pest species, so we move the rabbit response to the antecedent.

```
In [9]: draw_implication_network2(PCUList7,
                     ['posrats_rabbits', 'posrats_mice'],
                     'PCUList7_1', niceNames = None, controlSymbol = 'downarrow')
        os.system("dot -Tpng PCUList7_1.dot > PCUList7_1.png")
```

```
PCUList7_1.pdf has been created
```

```
Out[9]: 0
```



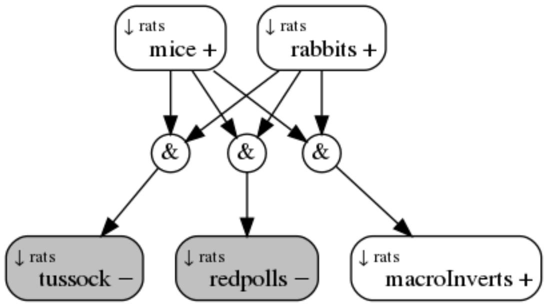
```
PCUList7_1
```

One could end the splitting of the remaining PCUs here. Alternatively, one could choose to emphasise the contingency upon the response of redpolls, by further splitting the network.

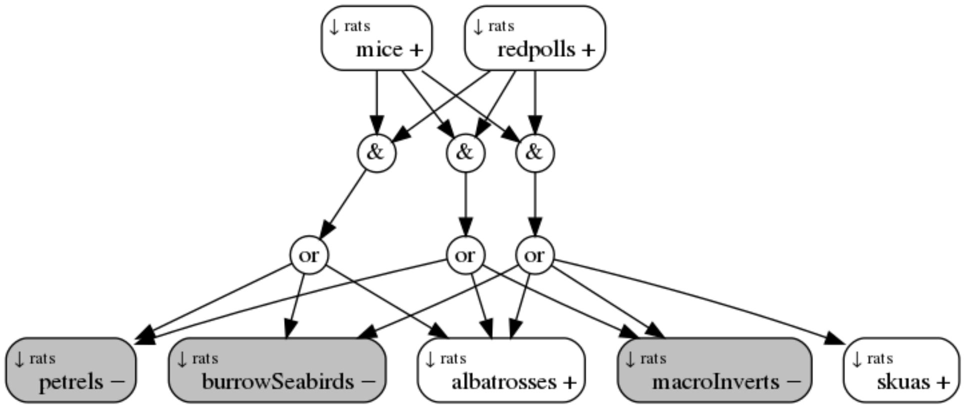
```
In [10]: PCUList8 = list() # for rabbit and mice relationships
         PCUList9 = list() # the remainder, which all involve redpolls

         for PCU in PCUList7:
             if 'posrats_rabbits' in PCU:
                 PCUList8.append(PCU)
             else:
                 PCUList9.append(PCU)

         draw_implication_network2(PCUList8,
                      ['posrats_rabbits', 'posrats_mice'],
                      'PCUList8_1', niceNames = None, controlSymbol = 'downarrow')
         os.system("dot -Tpng PCUList8_1.dot > PCUList8_1.png")

         draw_implication_network2(PCUList9,
                      ['posrats_redpolls', 'posrats_mice'],
                      'PCUList9_1', niceNames = None, controlSymbol = 'downarrow')
         os.system("dot -Tpng PCUList9_1.dot > PCUList9_1.png")
```

```
PCUList8_1.pdf has been created
PCUList9_1.pdf has been created
```

```
Out[10]: 0
```



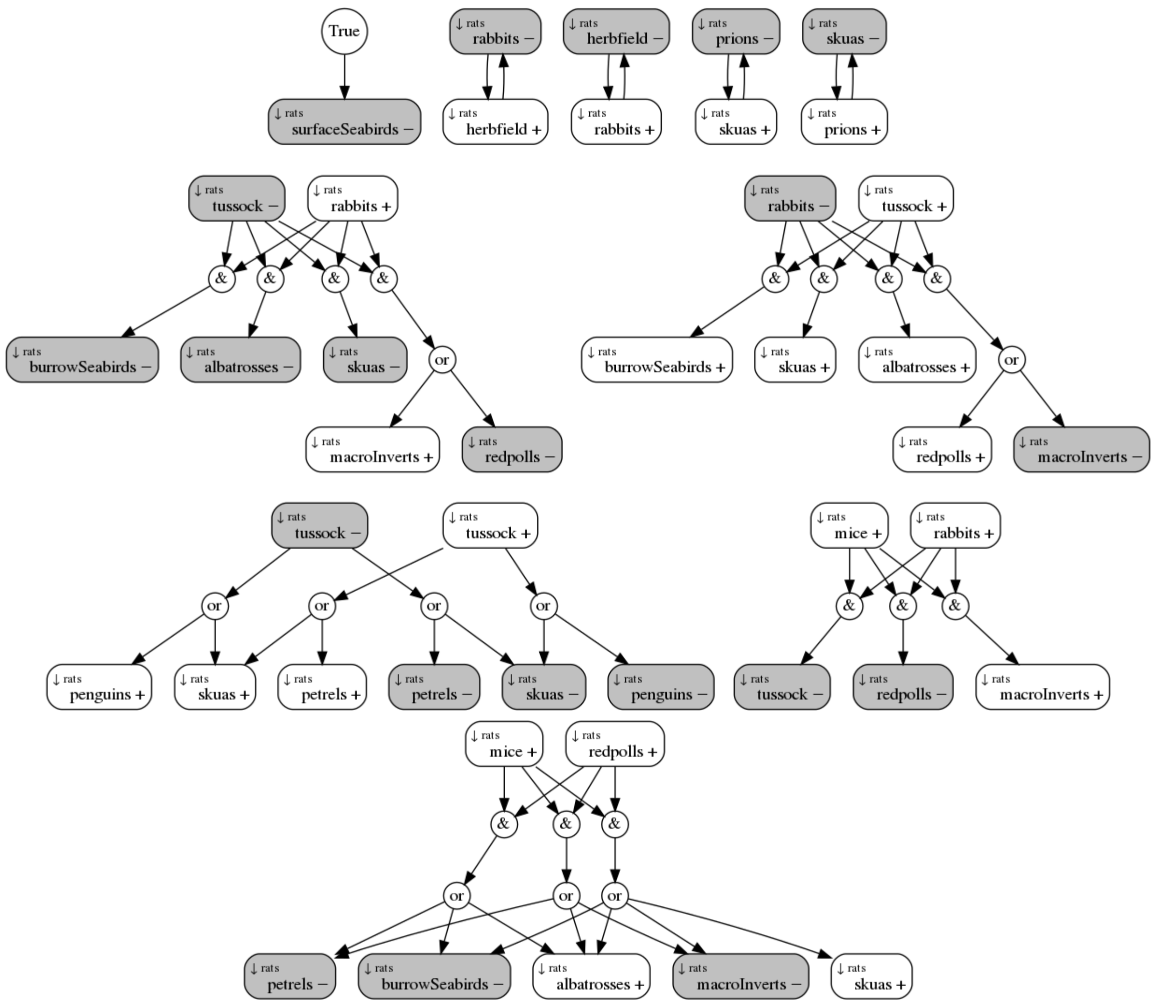
```
PCUList8_1
```



```
PCUList9_1
```

Potential final presentation

The presentation below combines all of the subnetworks that we split above. The decisions that were made to produce this network were somewhat arbitrary, and the best choice depends upon the needs of conservation decision-makers, and may also reflect our causal understanding of the relationships between species.



C.4 Applying the Boolean approach to an arbitrary model

The purpose of this tutorial is to show the Boolean analysis might be applied to an arbitrary model. The (fictitious) model concerns the social and environmental outcomes of river management.

This Jupyter Notebook is archived as:
Tutorial-4-Using_the_Boolean_approach_on_your_model.ipynb

Notebook

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
         from findpcu import intList2boolexpr, boolexpr2RespvalList

         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 responseCombination]), 2)
                                       for responseCombination in collectedResponses ]
         observedInts
         unobservedInts = set(range(2**len(responsesList)))
         # ... and loop through the observed response combinations,
         # discarding those that were observed

         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_lei, ~int_qua),
    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_lei, ~int_qua),
    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_lei, ~int_qua),
    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_lei, ~int_qua),
    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_lei, ~int_qua),
    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_qua),
    And(int_eng, ~int_sat, int_pri),
    And(int_eng, ~int_sat, ~int_qua))
```

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

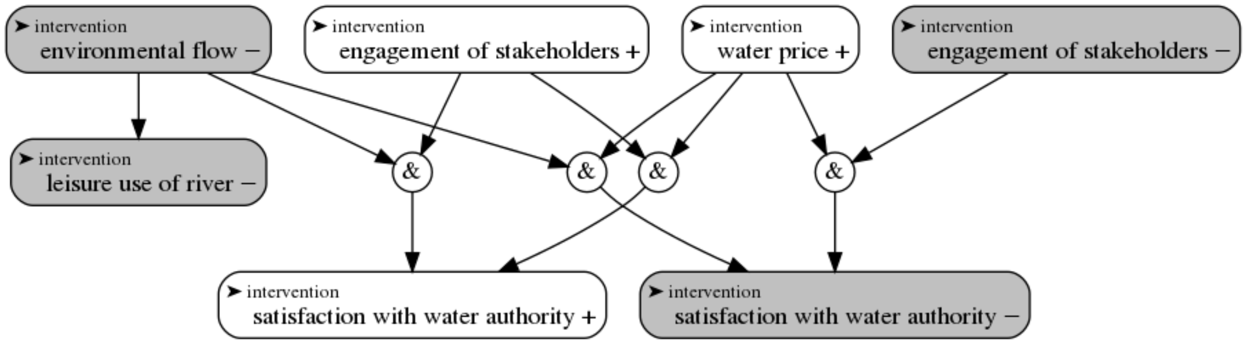
```
In [61]: PCUList = boolexpr2RespvalList(boolExprMin, x2s)
         PCUList

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', 'posint_pri', 'negint_pri'],
'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`.

D Interpreting the results of the Boolean approach

D.1 Boolean variables and logical implications

A Boolean variable is a variable having one of only two values, typically the truth values, True or False. For example, a Boolean variable ‘raining’ can may take the value: True, if it is raining; or False, if it is not.

Boolean variables can be linked by implications to create conditional statements. For example:

Statement 1: ‘if it is raining then the streets are wet’

links Boolean variables ‘raining’ and ‘wet streets’ by an implication (‘then’). This can be written as a logical implication statement:

$$\text{raining} \;\rightarrow\; \text{wet streets} \tag{D.20a}$$

where the right-arrow plays the role of the word ‘then’.

It is worth noting that the converse of a true statement is not necessarily true, i.e. the direction of an implication cannot be ‘reversed’. For example, it does not follow from Statement 1 that ‘if the streets are wet then it is raining’; the streets might be wet for some other reason, e.g. because the street cleaner has come past.

The contrapositive of a true statement is also true. For example Statement D.20a can be rearranged to say:

Statement 1b: ‘if the streets are not wet then it is not raining’

which can be written

$$\text{not wet streets} \;\rightarrow\; \text{not raining} \tag{D.20b}$$

where ‘not’ is an operator that negates the value of a Boolean variable, i.e. switches from False to True, and vice versa.

The example in Eq. D.20b also highlights a key difference between the implicit meaning of an “if . . . then . . .” in an English sentence and a logical implication: implications are statements about truth, not causality. Rain causes the streets to be wet. The absence of wet streets does not cause the absence of rain. But it is true that, if the streets are not wet, then it is not raining.

The operators ‘and’ and ‘or’, written as $\wedge$ and $\vee$ respectively, can be used to connect Boolean variables to create more complex conditional statements. For example:

Statement 2: ‘if it is raining and I do not have my umbrella then I will have wet hair’ may be written

$$\text{raining} \ \wedge \ \text{not have umbrella} \ \rightarrow \ \text{wet hair} \tag{D.21a}$$

The rules of Boolean algebra can be used to rearrange the statement into other equivalent forms. For example, the logical implication in Eq. D.21a can be rearranged to

$$\text{not wet hair} \ \rightarrow \ \text{not raining} \ \vee \ \text{have umbrella} \tag{D.21b}$$

$$\text{not have umbrella} \ \wedge \ \text{not wet hair} \ \rightarrow \ \text{not raining} \tag{D.21c}$$

and so on. In general, an implication with k terms can be rearranged into $2^k - 2$ different implications.

A convenient way to summarise a set of equivalent implication statements is to write the combination of Boolean variables that is impossible by that statement. For example, Eqs. D.20a and D.20b can be rearranged

$$\text{raining} \ \wedge \ \text{not wet streets} \ \rightarrow \ \text{False} \tag{D.22}$$

and Eqs. D.21a to D.21c can be rearranged

$$\text{raining} \ \wedge \ \text{not have umbrella} \ \wedge \ \text{not wet hair} \ \rightarrow \text{False} \tag{D.23}$$

The combinations of Boolean variables on the left-hand sides of Eqs. D.22 and D.23 are impossible combinations. These impossible combinations are analogous to the impossible species-response combinations, discussed in the body of the text, that form the PCUs.

In the body of the text, we have also discussed the logical implication network, which chains logical implications together by their shared terms. Fig. D.6 shows one possible logical implication network that results from combining the examples above.

D.2 Species responses as Boolean variables and model predictions as logical implications

In the case study in the body of the text, a particular species A's response to a pest control program can take one of two values: positive, which we denote by $A+$; or negative, which we denote by $A-$. Because the species responses are two-valued, we can treat them as a Boolean variable, by mapping the positive response to True, and mapping the negative response to False. By doing so, we are then able to harness the machinery of Boolean algebra, in order to analyse and summarise the species responses in the ecological model as a whole.

The goal of the Boolean approach is to find conditional statements that link the model's predictions about species responses together. For example, if through model simulations we find that:

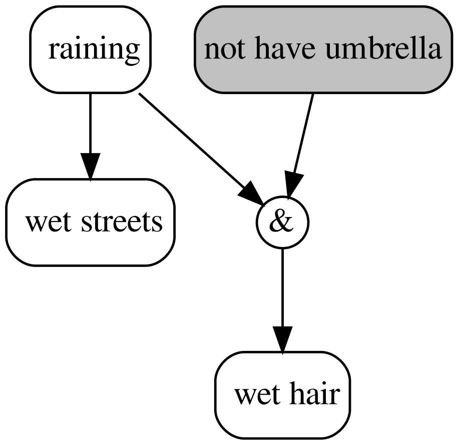


Figure D.6: A logical implication network that can be formed by combining the logical implications in Eq. D.20a and D.21a.

'if A responds positively then B responds positively'

then that can be written as a logical implication statement

$$A+ \rightarrow B+ \tag{D.24}$$

In contrast to the probabilistic QM approach, implication rules like Eq. D.24 are deterministic. If we say that Eq. D.24 is true in the model, then we are saying that the model always predicts that a positive response in species A will be accompanied by a positive response in species B. In the case study, the structure of the network of interactions between species is given, however the interaction strength values are unknown. Therefore Eq. D.24 can be more specifically interpreted as:

'in the model, if A responds positively then B responds positively, regardless of the interaction-strength values'

D.3 Logical implication networks from ecological models

In the following examples, we will assume that we have simulated models of pest suppression, similar the the Macquarie Island case study, and used the Boolean approach to analyse the results. We will assume that we have modelled the suppression of a pest species, P, and that we are interested in the responses of the other species in the network: A, B, C, etc.

The simplest possible outcome is that a single-species response is impossible in the model, regardless of what the other responses are. For example, if the model never predicts a negative response in A

$$A- \rightarrow \text{False} \tag{D.25}$$

then this can be represented by the implication network in Fig. D.7a. Because $A-$ is a Boolean variable, the logical implication can also be rearranged

$$\text{True} \rightarrow A+ \tag{D.26}$$

resulting in the implication network in Fig. D.7b. Fig. D.7b would be the more usual way of presenting the result, because it gives the result in terms of what the model predicts will happen to a species.

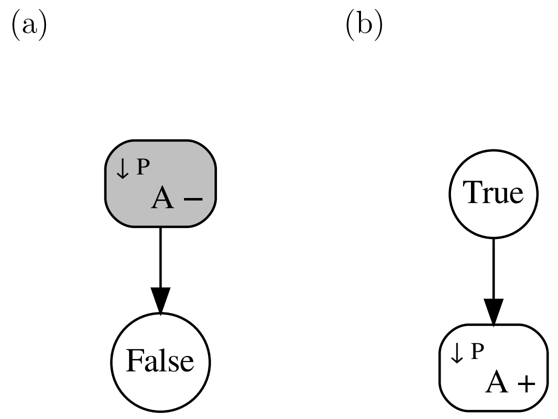


Figure D.7: Two logical implication networks corresponding to the model prediction that species A will never respond negatively to the suppression of pest-species P. A negative response in A is abbreviated 'A−', a positive response in A is abbreviated 'A+', and the '↓P' in the top left-hand corner of each node reminds us that the context is species responses to the suppression of P.

The kind of single-species response scenario that is shown in Fig. D.7 is easily revealed by existing QM methods. A weighted-predictions matrix analysis would result in $W_{AP} = 1$. A Monte Carlo simulation will return a 100% probability that A responds positively. The advantage of the Boolean approach is that it can analyse more complex scenarios, where a species response is contingent upon the responses in other species.

The simplest contingent case involves two species connected by a unidirectional implication. For example, the logical implication network in Fig. D.8a contains one logical implication

$$A+ \rightarrow B- \tag{D.27a}$$

which reads as:

'if A responds positively then B responds negatively'

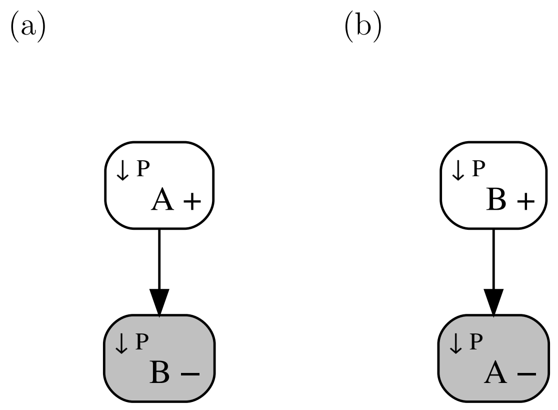


Figure D.8: Logical implication networks corresponding to the logical implication in Eq. D.27a and Eq. D.27b.

The logical implication in Eq. D.27a can also be rearranged to obtain its contrapositive:

$$B+ \to A- \tag{D.27b}$$

which reads as:

'if B responds positively then A responds negatively'

and is shown in Fig. D.8b.

Eq. D.27a and Eq. D.27b are equivalent, and are derived from a single PCU:

$$A+ \wedge B+ \to \text{False} \tag{D.27c}$$

In other words, the results from our model simulations never predict a simultaneously positive response in A and a positive response in B.

It is important to note that the direction of an implication cannot simply be reversed. For example, from Fig. D.8a, we cannot say *'if B responds negatively then A responds positively'*. In the full truth table corresponding to the PCU in Eq. D.27c (Table D.7), when B responds negatively then either a positive or a negative response in A is possible. The only valid way to reverse the order of the variables in the implication is to rearrange it using Boolean algebra; in this case, to take the contrapositive, shown in Fig. D.8b.

Rearranging the implications can be useful for obtaining statements that are more practical from a management perspective. For example, the implication shown in Fig. D.9a,

$$A+ \to B- \vee C- \tag{D.28}$$

can be rearranged into 5 other equivalent logical implications, with corresponding implication networks Fig. D.9b-f. The choice of which network to present to stakeholders depends

Table D.7: The truth table corresponding to Eq. D.27c, with arrows highlighting the rows corresponding to the situation in which B responds negatively. When B responds negatively, either a positive or a negative response in species A is possible.

Suppression of P causes positive response in: A	B	Species-response combination impossible in the model?	
False	False	False	←
False	True	False	
True	False	False	←
True	True	True	

upon practical considerations. For example, if species A and B are of particular conservation concern, then Fig. D.9a could be rearranged to produce Fig. D.9c, which frames the result in terms of impacts on A and B. As another example, if monitoring species B and C is particularly easy, then Fig. D.9e may be useful to managers in the design of a post-intervention monitoring programme, to provide early warning about potential deleterious effects on the more obscure species A.

It is important to note that the 'or' operator in an implication network is inclusive. Inclusive means that only one of the consequents has to occur, but more than one may also occur. For example, in Fig. D.9a, a positive response in A predicts a negative response in B or a negative response in C. Therefore the following three outcomes could occur: (1) $B-$ and $C+$, (2) $B+$ and $C-$, or (2) $B-$ and $C-$.

An 'and' operator requires all antecedents to be satisfied in order to *guarantee* the consequent. For example, in Fig. D.9d, the model only guarantees a negative response in species C if there is a positive response in both A and B. It is *possible* that species C might respond negatively if only A or only B is positive (see Fig. D.9a-b), however it is not guaranteed.

The absence of a species from the implication network indicates that any response in that species is possible, and that which response occurs cannot be predicted from the other species responses. For example, if the result shown in Fig. D.9 was obtained for a four-species ecosystem, with species A, . . ., D, then any response in species D is possible.

For more complicated implication networks, the basic interpretation depends upon interpreting each of the implications that appear in the subnetworks within it (as above). Subnetworks can be identified by finding species responses that are joined by a direct implication, or joined to the same 'and' or 'or' node and by implications. For example, in the implication network in Fig. D.10, each of the subnetworks (circled in a different colour) corresponds to a PCU as follows:

$$A+ \rightarrow \text{False} \tag{D.29a}$$

$$B- \rightarrow \text{False} \tag{D.29b}$$

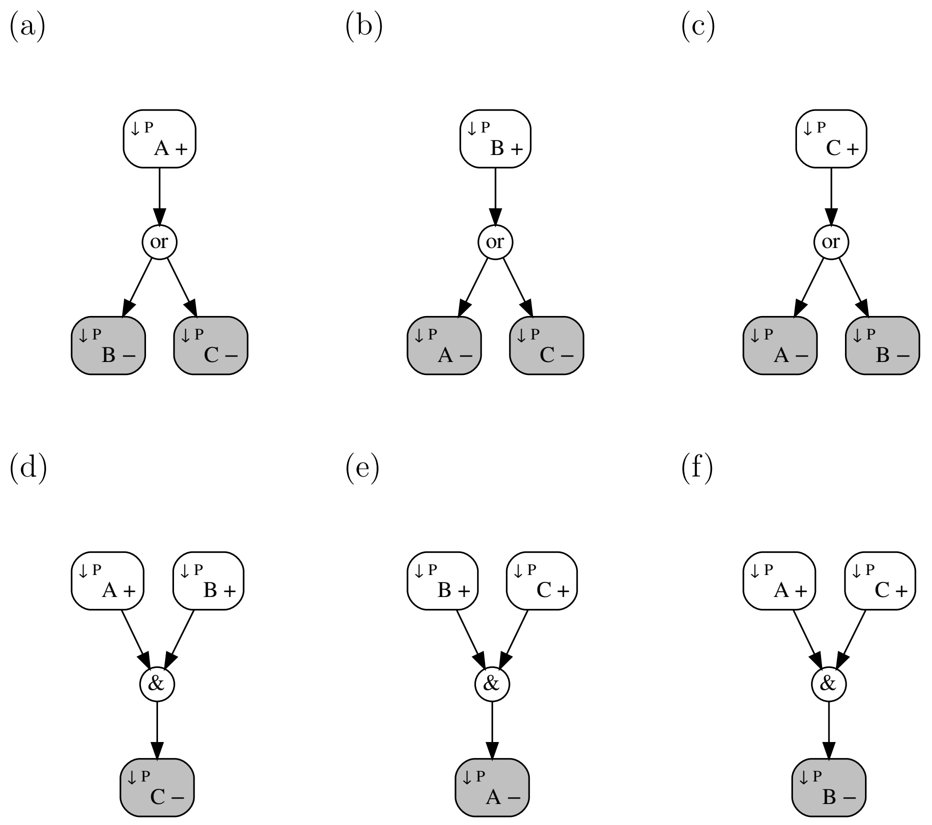


Figure D.9: The logical implication network corresponding to Eq. D.28 (a), and five equivalent networks that can be derived from it (b-f).

$$C+ \wedge D- \rightarrow \text{False} \tag{D.29c}$$

$$C+ \wedge E+ \wedge F- \rightarrow \text{False} \tag{D.29d}$$

$$C+ \wedge G- \wedge H- \rightarrow \text{False} \tag{D.29e}$$

$$G- \wedge I+ \wedge J- \wedge K+ \rightarrow \text{False} \tag{D.29f}$$

For large implication networks, it may take some work to separate and identify useful subnetworks. The choices one makes depend heavily upon the conservation decision-making context, and the tutorial in SI C.3 provides an example of how these decisions may be made.

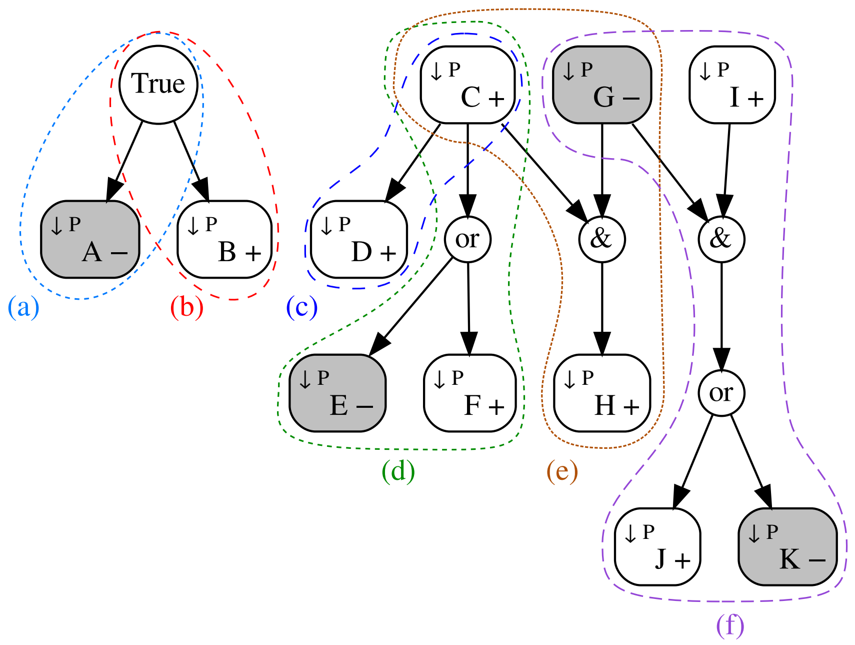


Figure D.10: A logical implication network, highlighting the subnetworks corresponding to each implication derived from the PCUs in Eqs. D.29a to D.29f.

In more complicated implication networks, information can also be gleaned from the relationships between subnetworks. Disconnected subnetworks indicate that the species within those subnetworks respond together in some way, and can often be related to the appearance of tightly connected subnetworks in the original species interaction network. However, the implication network can also identify relationships between species responses that are not obvious from analysing original interaction network itself; this reflects the complex-systems nature of ecosystem behaviour, which results from direct and indirect feedbacks between species.

Species responses that appear in multiple subnetworks may indicate key species, or determinants of the system's behaviour (e.g. $C+$ in Fig. D.10). This may be interesting from a theoretical perspective, to identify important species in the system, or from a practical perspective, if those species are candidates for a post-control monitoring programme. However it is important to remember that implication relationships are statements about truth, not causality. Again using the example in Fig. D.10, one cannot say that a positive response in C 'causes' a positive response in D; the cause of their relationship involves all of the direct and indirect feedbacks between those two species and other species in the system.

A particularly interesting case, from a pest-control perspective, is when another pest's response is highly connected in the logical implication network. For example, in Fig D.11, the mesopredator M's response to the control programme has a key determining effect upon outcomes on other species. This situation suggests that a multi-species control programme, where both the pest P and the mesopredator M are suppressed, could have beneficial outcomes. A modeller should then perform further simulations, to determine what the model predicts that the outcomes of a multi-species control programme will be.

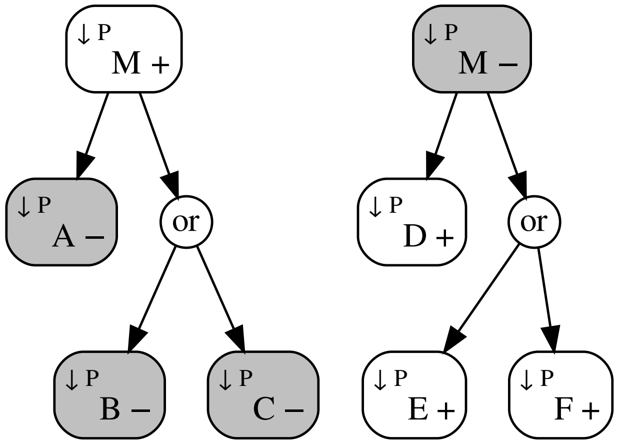


Figure D.11: A logical implication network where a mesopredator species M has a key determining effect upon species outcomes.

The difficulty of interpreting an implication network increases with the number of species involved in each subnetwork. For example, in Fig. D.12, a positive response in A does not give us much information about what will happen to the other species. Because the 'or' node is inclusive, a positive response in A may be accompanied by the response shown for: B only; B and C only; B, C, and D only; etc. While it is difficult to interpret such a result in terms of predictions, it has some use in that it indicates that the species responses are highly (though not completely) indeterminate.

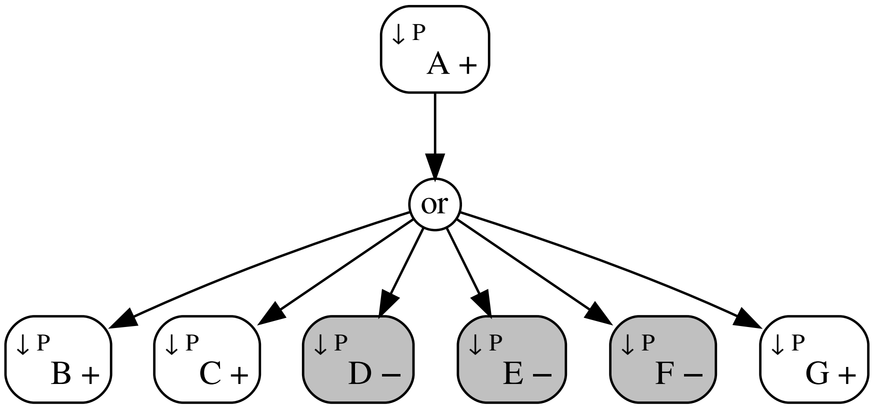


Figure D.12: A logical implication subnetwork containing many species responses reflects indeterminacy in those species responses.

E Additional results

The sampling method influenced how easy it was to obtain a community matrix that passed the plausibility constraints (Fig. E.13). Increasing the strength of the stability constraint increased the difficulty obtaining a plausible community matrix. When the sampling was performed on the community matrix parameters (Raymond baseline), approximately 0.5 matrices were rejected before a stable matrix was found. In contrast, when sampling on the Lotka-Volterra parameters (test 1) an average of approximately 950,000 matrices were rejected before the plausibility constraints were met (cf. Christianou and Kokkoris, 2008).

Additional species-response predictions, that were not included in the body of the text, are found in Fig. E.14 and E.15.

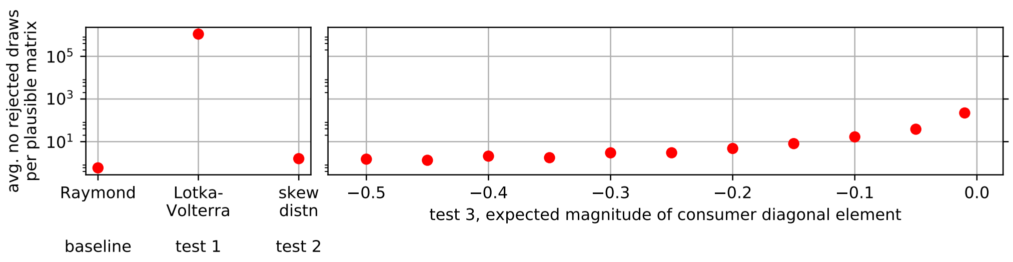


Figure E.13: Additional results from the different Monte Carlo simulation methods that were applied to the Macquarie Island case study.

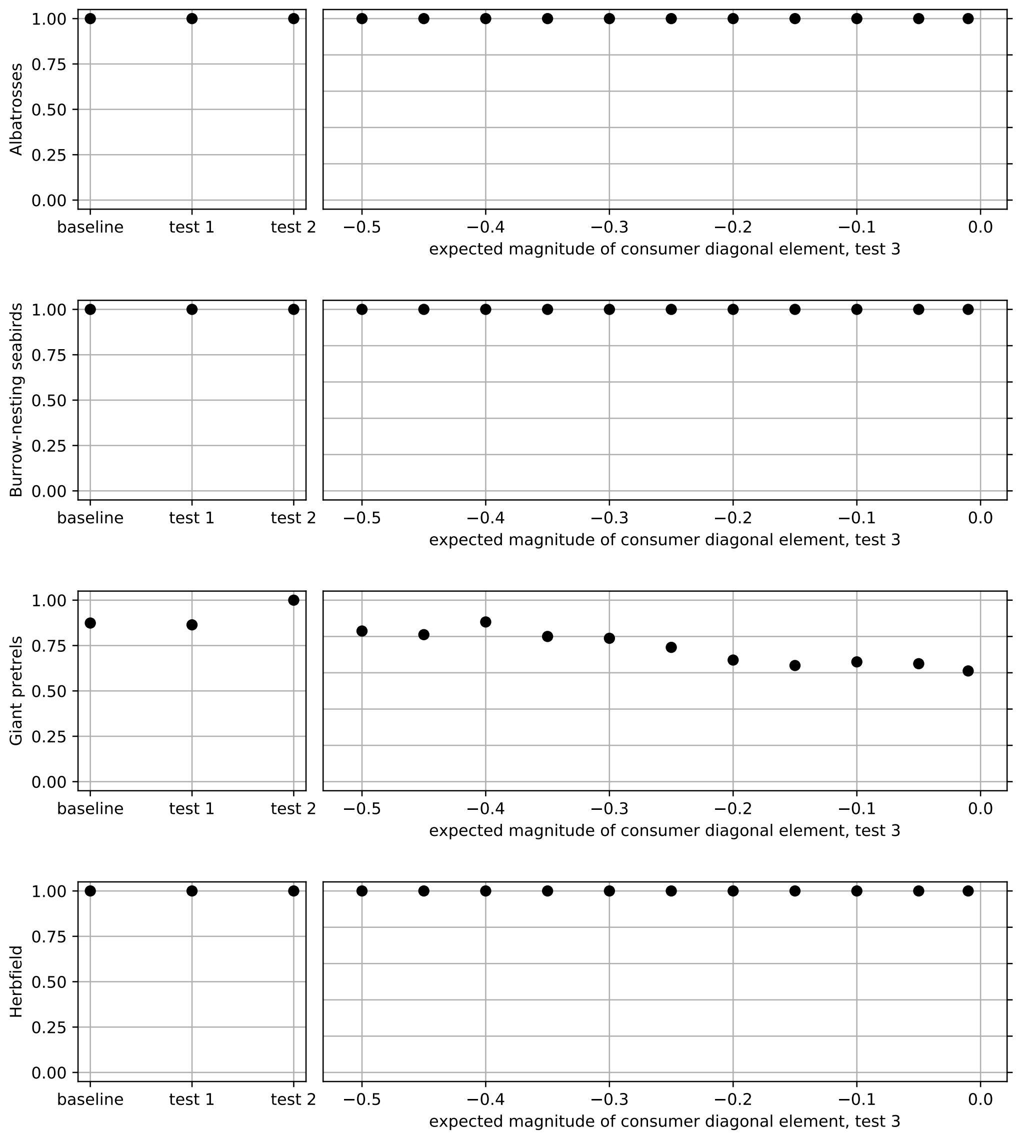


Figure E.14: Additional results from the different Monte Carlo simulation methods that were applied to the Macquarie Island case study.

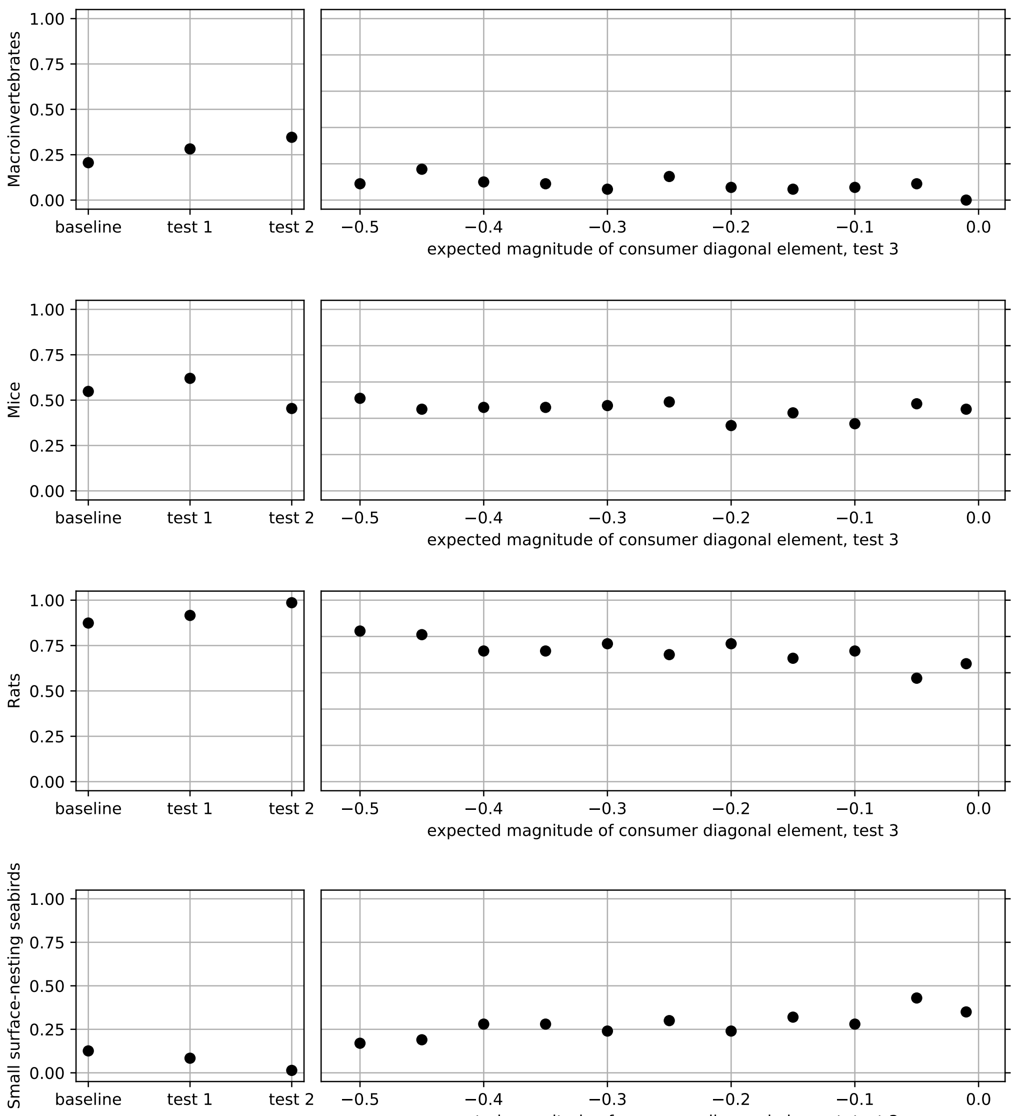


Figure E.15: Additional results from the different Monte Carlo simulation methods that were applied to the Macquarie Island case study.

F Multi-species press-perturbation

The response of a species to a multi-species pest-control was modelled in Raymond et al. (2011) by summing the corresponding elements in the sensitivity matrix (see also Ramos-Jiliberto et al., 2012; Harvey et al., 2016). In this appendix, we show that this implicitly assumes that the magnitude of the perturbation on each pest is equal.

In the case of a single-species perturbation, the effect on species i of a press-perturbation of species j is found by rearranging Eq. 12 of Nakajima (1992):

$$dx_i = s_{i,j} dz_j \tag{F.30}$$

where dx_i is the change in the steady-state population size of species i or the focal species' response, dz_j is the rate in inflow or removal of species j, and $s_{i,j}$ is the corresponding element of the sensitivity matrix. In this case, the sign of the species response dx_i in Eq. F.30 can be determined without knowing the magnitude of the perturbation (assumed small).

In the case of a multi-species perturbation, the effect on species i of simultaneous press-perturbation of a set of species $j \in P$ is found similar to Eq. 15 in Nakajima (1992):

$$dx_i = \sum_{j \in P} s_{i,j} dz_j. \tag{F.31}$$

In this case, if there are a mix of signs of elements $s_{i,j}$ in the sum in Eq. F.31, then the sign of the species response dx_i cannot be determined without knowing the relative magnitudes of the perturbations dz_j.

When Raymond et al. (2011) summed the elements of the sensitivity matrix to obtain the sign of the species' responses to a multi-species press-perturbation, they were effectively assuming that all dz_j in Eq. F.31 were equal in magnitude. Alternatively, they may have been invoking an implicit assumption about the probability distribution of the relative magnitudes of dz_j.