Supplementary Material to: Robust network inference using response logic

Torsten Gross[1,2,3], Matthew Wongchenko[4], Yibing Yan[4], and Nils Blüthgen[1,2,3]

[1]Charité - Universitätsmedizin Berlin, Institut für Pathologie, Berlin, Germany
[2]IRI Life Sciences, Humboldt University, Berlin, Germany
[3]Berlin Institute of Health, Berlin, Germany
[4]Genentech Inc., Oncology Biomarker Development, USA

S1 Encoding the response logic in ASP

The response logic is formulated in the framework of Answer Set Programming. We use the syntax of the input language gringo. A straightforward introduction is provided in section 4 of (Videla et al., 2015). The following facts and clauses constitute the logic program.

At the beginning, the program needs to state the facts, i.e. the number of nodes in the network as constant `n`, the set of perturbations as functions `pert(I,OUT)`, where variable `I` names a specific perturbation and variable `OUT` indicates the node(s) targeted by the perturbation (could be multiple), as well as the rectified response pattern as functions `response(I,N)` and `-response(I,N)` (classical negation), where variable `I` names a specific perturbation and variable `N` indicates the node(s) responding or not responding to the perturbation, respectively. If a perturbation is defined to target multiple nodes, it is assumed to cause a response on all nodes that are path-connected to any of the perturbed nodes. Strictly speaking, such a response no longer corresponds to a transitive closure, as it describes a node's reachability from any of multiple nodes. But for the sake of simplicity, we will nevertheless keep the terminology.

The program then follows the generate-define-test pattern, as below. The function `tc(I,IN)` represents the transitive closure of the answer set network.

```
%%%Generate%%%
node(0..n-1).
{edge(OUT,IN)} :- node(IN), node(OUT).

%%%Define%%%
tc(I,IN) :- edge(OUT,IN), pert(I,OUT).
tc(I,IN) :- edge(OUT,IN), tc(I,OUT).

%%%Test%%%
```

```
:- not tc(I,N), response(I,N).
:- tc(I,N), -response(I,N).
```

The last two lines are the integrity constraints that discard any answer set whose transitive closure is in disagreement with the rectified response pattern. ASPs distinction between classical ("-") and default ("not") negation allows to only test the integrity for the known entries of the response pattern and to ignore the test for any entries that are not known.

For large networks the search space of the above program becomes very large with negative effects on performance. This is especially problematic during the rectification of the response pattern because in principle every entry of the response pattern must be tested for consistency. We implemented two strategies to speed up the process. First, at each step of the iteration, we check if the previous transitive closure is in agreement with the new data point. If so, we know that there is a conforming network and can accept the data point without an explicit test. Second, if there is no additional constraint (see below) stating the absence of a network edge, we can significantly simplify the logic program. In that case, if the logic program is satisfiable at all, the network with directly links from perturbed node to all its responding nodes, i.e. a network with star topology, constitutes a conforming network. As the rectification process only needs to determine satisfiability, a much more simple logic program avoids generating all possible networks but only tests the star network (which we do not need to state explicitly):

```
response(I,N) :- response(J,N), pert(J,M), J!=I,
                 response(I,M).
response(I,OUT) :- pert(I,IN), edge(IN,OUT).
response(I,N) :- response(J,N), pert(J,M), J!=I, pert(I,M).
```

The idea of this encoding is to define all additional responses that are implied by the star network from the given response pattern (line 1): if node N responds to a perturbation of node M (it is reachable from M) then node N will also respond to any other perturbation for which it is known that it causes a response at node M. This defines the minimal set of responding nodes consistent with the response logic. Any constraints on the absence of edges would imply (longer) non-direct paths that could potentially imply more nodes to respond to certain perturbations. This program does not even need to explicitly state an integrity constraint because an observed non-response given as classical negation `-response(I,N)`, already rules out that `response(I,N)` is contained in the (single) answer set. Furthermore, line 2 adds responses that are implied by edges that are known to exist and the last line is needed in the case where multiple perturbations hit the same node, but a perturbed node itself does not respond.

As discussed at length, we might wish to incorporate additional, external domain knowledge into the logic program. The most direct way of doing so is to simply state the knowledge about presence or absence of e.g. the link from node 2 to node 3 as a fact:

```
edge(2,3).
```

or

```
-edge(2,3).
```

respectively.

To encode bounds on the number of links that enter or leave one or a group of nodes is simple, due to ASPs built-in `#count` aggregates. If, for example, we were to maximally allow for 3 incoming edges to node `1` we would add the following integrity constraint to the logic program

```
:- #count {OUT : edge(OUT,1) } > 3.
```

Obviously, this statement could easily be adapted to formulate a lower bound, or bounds for groups of nodes.

It is more complex to state the parsimony constraint that filters out any network for which the removal of any link does not change the transitive closure. Essentially, for a given network, we define a set of links, `x_edge(OUT,IN)` that are subject to removal. Those are all the edges that are part of the network but which are not enforced by external knowledge. Then, the idea is to define all networks with a link removed, determine their transitive closure and compare it to that of the original network. The final integrity constraint removes the answer set if any of the reduced networks does not have a reduced (that is, it has the same) transitive closure as the original one.

```
r_edge(OUT,IN,NOUT,NIN) :- (OUT,IN)!=(NOUT,NIN),
                           edge(OUT,IN), x_edge(NOUT,NIN).
r_tc(I,IN,NOUT,NIN) :- r_edge(OUT,IN,NOUT,NIN), pert(I,OUT).
r_tc(I,IN,NOUT,NIN) :- r_edge(OUT,IN,NOUT,NIN),
                       r_tc(I,OUT,NOUT,NIN).
has_reduced_tc(NOUT,NIN) :- not r_tc(I,IN,NOUT,NIN),
                            tc(I,IN), x_edge(NOUT,NIN).
:- not has_reduced_tc(NOUT,NIN), candidate_edge(NOUT,NIN).
```

As this logic creates a significant amount of additional variables, we observed that the parsimony constraint will suffer performance issues when applied to more than a few tens of nodes.

Finally, a crucial feature of the response logic approach is the ability to generate the union of all conforming networks, which points out the links that can or cannot be uniquely determined. Typically, the number of conforming networks is intractable which precludes to simply enumerate all solutions and then compute their union directly. Rather, we let a logic program find it directly. The union of all answer sets of an ASP program is also termed brave consequences, hence the naming conventions below. In a first step, we need to annotate the presence or absence of an edge explicitly.

```
edge_brave(OUT,IN,1):-edge(OUT,IN).
edge_brave(OUT,IN,0):-not edge(OUT,IN),node(OUT),node(IN).
```

Then, the idea is to repeatedly solve the logic program, while iteratively building up the union of conforming networks. That is, we record for each edge whether the solutions generated so far, found it to be always absent, always present or neither (sometimes present, sometimes absent). To obtain the union over all answer sets, the program is further constrained with every new solve call to only permit networks that are not a subset of the union that was recorded so far. This is encoded by the following integrity constraint.

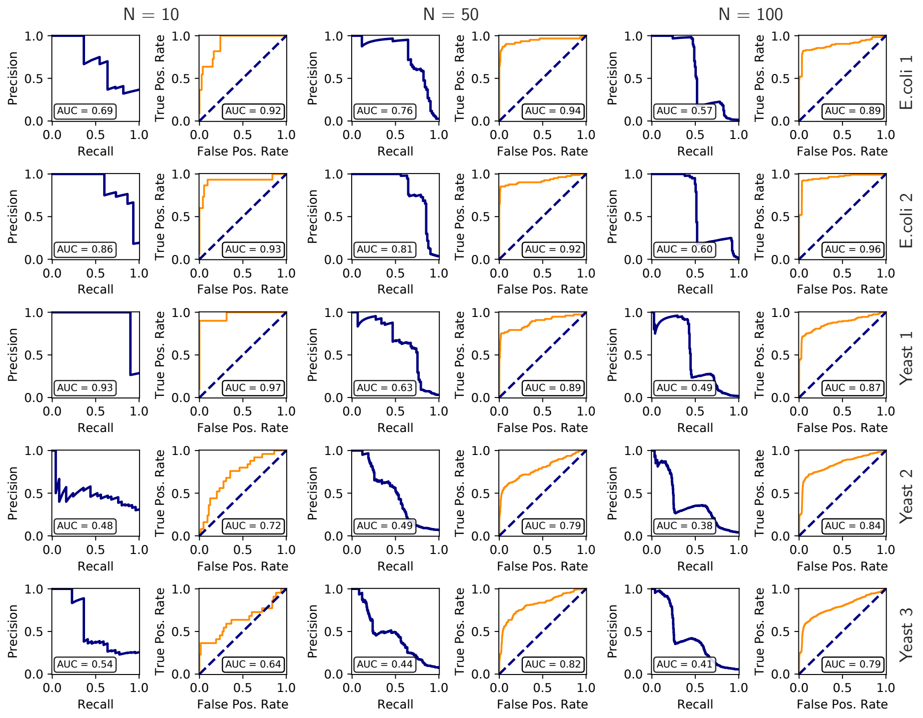

Figure S1: DREAM3 challenge 4: Precision Recall (blue) and ROC (red) curves for the response logic predictions of the five networks.

```
:- not edge_brave(OUT,IN,B) : node(OUT), node(IN), B=0..1,
                              @is_in_union(OUT,IN,B) == 0.
```

The `@is_in_union(OUT,IN,B)` construct is a function called by the solver that will return 1 if the presence or absence (`B` is 1 or 0, respectively) of edge `OUT` to `IN` is already recorded in the union and 0 otherwise. Thus, every solve call adds new elements to the union until there are no more conforming networks that show the presence or absence of a link, beyond what is already represented in the union. This is when the program is no longer satisfiable and the repeated solve calls stop.

S2 Additional information about the inference of DREAM in-silico networks

The rectification of the response pattern requires confidence scores, C. Those are defined as a normalized distance of each entry of the global response matrix R to 1, which was chosen as the response threshold. The normalization is chosen such that confidence levels range from zero, for $R_{ij} = 1$, to one, for R_{ij} taking either the maximal $R^{\max}$ or minimal $R^{\min}$ value of all entries. Formally,

$$C_{ij} = \begin{cases} |R_{ij} - 1|/|R^{\max} - 1| & \text{for } R_{ij} - 1 > 0 \\ |R_{ij} - 1|/|R^{\min} - 1| & \text{for } R_{ij} - 1 < 0. \end{cases}$$

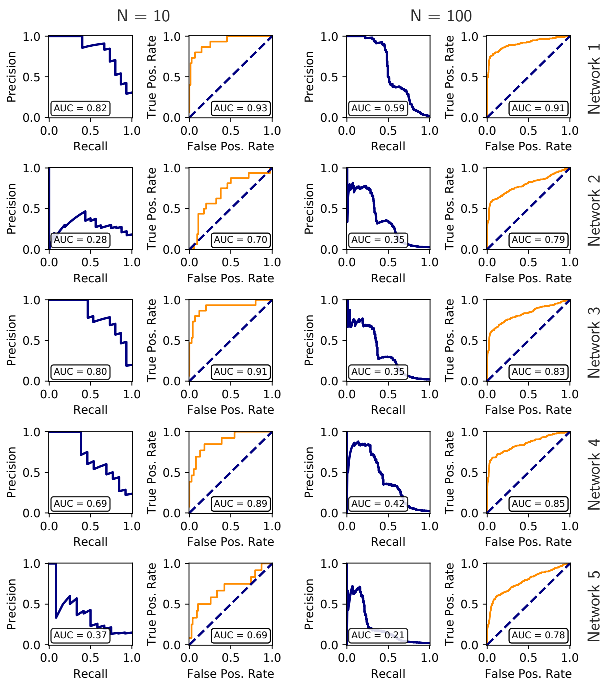

Figure S2: DREAM4 challenge 2: Precision Recall (blue) and ROC (red) curves for the response logic predictions of the five networks.

Both DREAM3 and DREAM4 provided five network challenges per network size. The precision-recall curve and the ROC curve for each network prediction are shown in Figure S1 and Figure S2.

The official DREAMTools package (Cokelaer et al., 2016) was used to compute the final prediction scores that are shown in main text Figure 4A. The scoring is based on p-values for the ROC and precision-recall area under the curves (AUC) that are computed from probability distributions of random network predictions. In the case of the DREAM3, N=100, Yeast3 network the precision-recall AUC of the response logic prediction is larger than that of any of the provided random network predictions. Therefore the determined p-value becomes zero and the overall network score becomes infinity. To obtain a more reasonable score we decided to modify the p-value computation in that case. We identified the largest AUC for which the random probability distribution shows a nonzero probability and simply defined an extended distribution whose probability decreases linearly from this point to zero probability at AUC=1 and computed the p-value based on this approximated distribution. This only concerned one of the five scored networks in the DREAM3, N=100 prediction and the resulting overall score, 139.7, is shown in main text Figure 4A, third panel. Another strategy to cope with the problem is to remove the network altogether and compute the overall score (mean of geometric mean of p-values) only from the remaining four networks. This provides a lower bound for the overall score of 122.9, which is still clearly better than the score of any of the competitors and the naive approach.

S3 Additional information about the analysis of the SW-48 cell line

A reverse phase protein array (RPPA) perturbation experiment was carried out on the SW-48 parental, the SW-48 E545K and SW-48 H1047R PI3K mutant cell lines under full serum conditions. Antibodies were chosen to cover the activity (phosphorylation) of a range of kinases. Cells were perturbed by single small molecule inhibitors, by single growth factors or by a combination of one growth factor and one inhibitor. Measurements were carried out in 8 technical replicates.

In a first step readouts were filtered as follows. First, we removed all readouts whose signal remained within the technical noise level throughout the treatments, as this indicates a malfunctioning antibody. The readout had to be a component or a target of the MAPK or the PI3K pathway. To decrease redundancy, we removed very closely related readouts. This included readouts of functionally related phosphosites on the same kinase, or kinases that showed near-identical qualitative behaviour due to their proximity within the signalling pathway. This left us with 10 different readouts. Concerning the perturbations, we filtered out ineffective, or redundant inhibitors and ligands. This resulted in the data set depicted in Figure S3.

To use this data in the response logic framework we need to convert it to a response pattern with according confidence scores. First, we need to localize the targets of the perturbations. As not all the direct targets of each perturbation were part of the readouts, we replaced those by the ones that were the closest downstream the signalling chain. Concerning HGF stimulation, we observed a strong response by the PDGF receptor. C-Met not being amongst the readouts, we chose the target accordingly. Lacking additional data, it is subject to speculation whether this behaviour could be explained e.g. by receptor co-activation (Xu and Huang, 2010) or impurities of the used ligand. This resulted in the following allocations.

perturbation	EGFRi	MEKi	AKTi	EGF	HGF	IGF
target	EGFR	ERK	AKT	EGFR	PDGFRB	AKT

This makes apparent that the perturbations only have four different targets (PDGFRB, EGFR, ERK, AKT).

Next, we need to determine the response behaviour with respect to perturbations of those four targets. Inhibiting an unstimulated signalling pathway can lead to saturation effects, when additional reduction of kinase activity is not possible. Thus, to faithfully track the inhibition response we decided to investigate inhibition while cells are stimulated, that is to compare ligand + inhibitor to only ligand. To ensure that such a stimulation actually affects the inhibited pathways, we chose EGF stimulation for the EGFR and MEK inhibitors and IGF stimulation for the AKTi inhibitor. The stimulation effects do not suffer from such saturation effects and were thus compared to basal levels (DMSO+PBS) directly. We compiled an overview of the resulting perturbation comparisons in Figure S4.

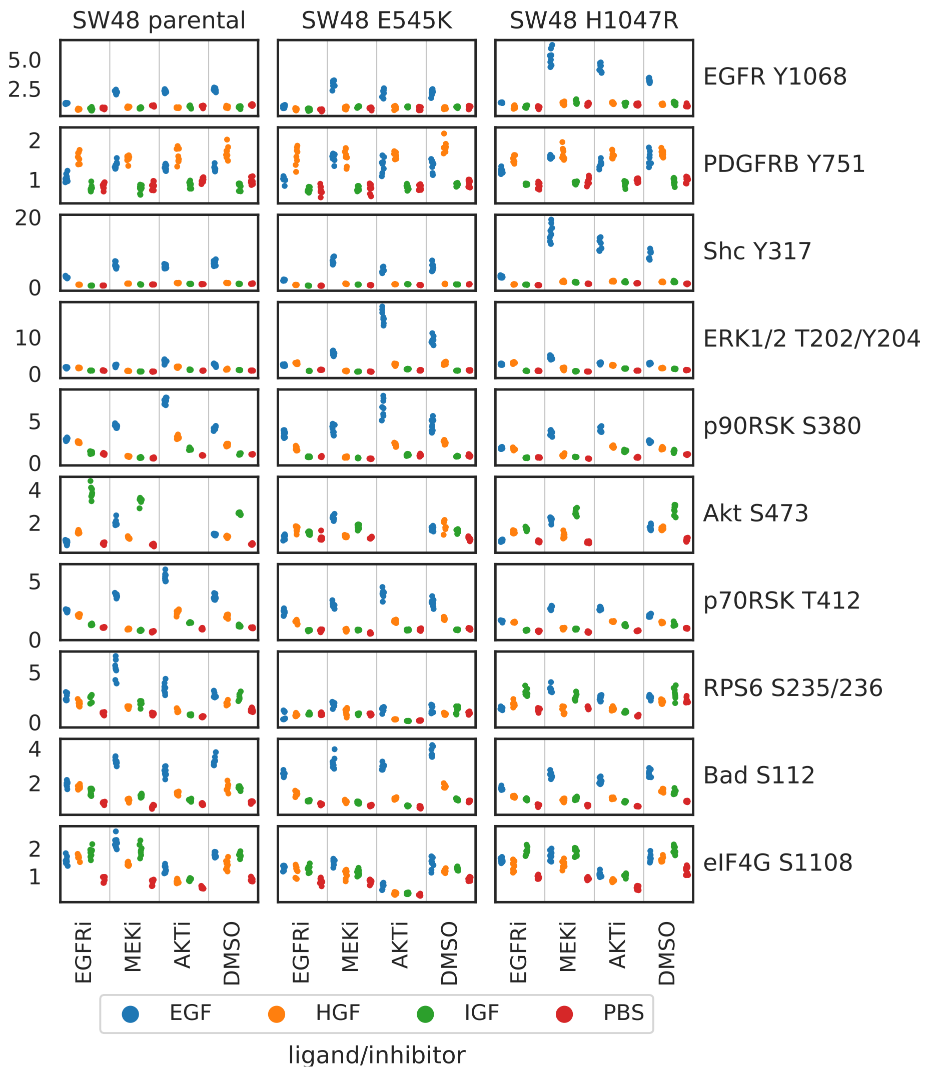

Figure S3: RPPA measurements on SW48 cell lines

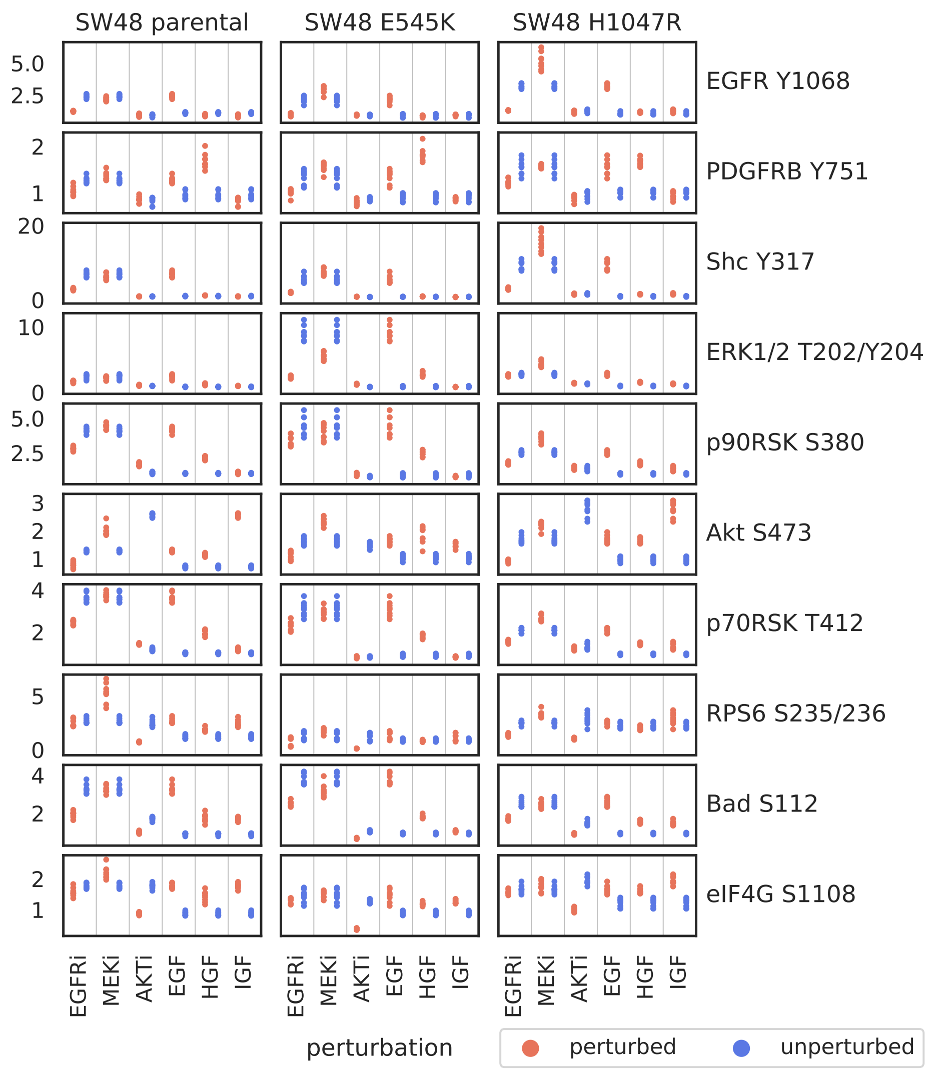

Figure S4: Same data as in Figure S3, but filtered and regrouped to highlight perturbation effects.

To decide whether a perturbation caused a response, we computed p values for each pair of perturbed-unperturbed samples using an unpaired, two-sample t-test. As the data replicates are technical in nature we chose a conservative significance threshold of 0.01. To compute confidence scores we used the same procedure as for the DREAM data, section S2, that is, the confidence scores represent a normalized distance between the p-value and the p-value threshold. Recall, that the six perturbations only have four different targets. To remove redundancy in the response logic sense, we decided to remove the two redundant data points with lower confidence (per cell line and readout). This resulted in the response patterns shown in main text Figure 5A and 6A.

The computation of the area under ROC and precision recall-curves in main text Figure 5A requires to rank the predicted links. However, depending on how the response logic approach is applied, some links can end up with the same score. We therefore, computed the PR AUC and ROC AUC as a mean over 100 AUC values that were generated by randomly reshuffling groups of links with equal score. Repeated computation of these PR AUC and ROC AUC values only varied in the third decimal place.

References

Cokelaer T. et al. (2016) Dreamtools: a python package for scoring collaborative challenges [version 2; referees: 1 approved, 2 approved with reservations]. *F1000Research*, 4.

Videla S. et al. (2015) Learning Boolean logic models of signaling networks with ASP. *Theoretical Computer Science*, 599, 79–101.

Xu A.M. and Huang P.H. (2010) Receptor tyrosine kinase coactivation networks in cancer. *Cancer Research*, 70, 3857–3860.