Mathematical Framework for Driver Extraction and Boolean Path Analysis using BooLEVARD

In this document, we present a concise, step-by-step framework for quantifying causal influence in a Boolean model. Beginning with the network's Boolean equations, we (1) compute their canonical DNFs (and the DNFs of their negations), (2) extract the drivers at a chosen stable state by selecting only the prime implicants that evaluate to true , and optimize those drivers via iterative substitution of unit-literal dependencies, and finally (4) enumerate all non-looping Boolean paths from designated input nodes to the target node. The resulting signed path count captures both the number of independent routes by which inputs can activate the output and the direction (activation vs. inhibition) of their influence.

1. Canonical DNF and NDNF

For each Boolean update function $f(x_i, \dots , x_n)$, we compute its canonical disjunctive normal form (cDNF) and the cDNF of its negation (cNDNF), as shown in Eq. (1-2):

$$cDNF(f)(x_i, \dots , x_n) = \bigvee_{m=1}^{M_i} \left(\bigwedge_{j:x_j \in X_m} L_j \right) \tag{1}$$

$$cNDNF(f)(x_i, \dots , x_n) = \bigvee_{n=1}^{N_i} \left(\bigwedge_{j:x_j \in X_n} L_j \right) \tag{2}$$

Here:

- M (resp. N) is the total number of prime implicants of f (resp. $\neg f$).
- X indexes the variables x_j appearing in the mth (resp. nth) prime implicant.
- $L_j \in \{x_j, \neg x_j\}$ denotes the literal for variable j in that implicant.

2. Driver extraction and optimization by unit-lateral propagation

Given a stable state $s \in \{0,1\}^n$, we extract for each node i the local driver $D(s)$ by selecting only those implicants that evaluate to true under s, then discarding literal polarity, as elucidated in Eq. (3):

$$D(s) = \begin{cases} \displaystyle\bigvee_{\substack{1 < m \leq M_i \\ \forall\, j:x_j \in X_m:\, L_j(s)=1}} \left(\bigwedge_{j:x_j \in X_m} x_j \right), & s_i = 1, \\ \displaystyle\bigvee_{\substack{1 < n \leq N_i \\ \forall\, j:x_j \in X_n:\, L_j(s)=1}} \left(\bigwedge_{j:x_j \in X_n} x_j \right), & s_i = 0. \end{cases} \tag{3}$$

Each selected implicants contributes a conjunction of node indices x_j, stripped of negations.

We iteratively substitute any driver that has collapsed to a single literal into its dependents. Let $\widehat{D_i}^{(0)} \coloneqq D_i(s)$ for node i, and define for $k \geq 0$ iterations:

$$\widehat{D_i}^{(k+1)} = \widehat{D_i}^{(k)} \left[x_j \mapsto \widehat{D_j}^{(k)}\right]_{j \in U^{(k)}}, U^{(k)} = \{j \colon \widehat{D_j}^{(k)}\ is\ a\ single\ literal\} \tag{4}$$

At each round, every variable x_j whose driver is already a unit literal is replaced by the dependencies of that literal in all other drivers, until no further substitutions occur.

3. Path elongation, counting, and signed scoring

From the fully optimized driver $\widehat{D_i}^*$, in Eq. (5) we build the set of all non-looping paths from node i to any entry node (those with unit-literal drivers, forming $Input$):

$$Paths_i = \begin{cases} \{(i)\}, & i \in Input, \\ \bigcup_{d=\{j_1,\dots,j_m\} \in D_i^*} \{(i)||p^{(1)}|| \dots || p^{(m)} \mid (p^{(1)}, \dots, p^{(m)}) \in Paths_{j_1} \times \dots \times Paths_{j_m}, i \notin p^{(k)}\ \forall k\} & i \notin Input. \end{cases} \tag{5}$$

Here each conjunctive block $d = \{j_1, \dots, j_m\}$ generates the Cartesian product of its child paths $Paths_{j_k}$; concatenating (i) yields every full path from i.

Next, in Eq. (6) we define the path count and attach a sign according to the node's state:

$$CP_i(s) = |Paths_i(s)|, v_i(s) = \begin{cases} +CP_i(s), & s_i = 1, \\ -CP_i(s), & s_i = 1. \end{cases} \tag{6}$$

Finally, averaging over all Z distinct stable states $s^{(1)}, \dots, s^{(Z)}$ in Eq. (7) yields the model-level score vector (available through the `collapsed=True` argument within the `CountPaths` function).

$$V_i = \frac{1}{Z} \sum_{z=1}^{Z} v_i(s^{(z)}) \tag{7}$$

Toy example

We consider a Boolean model (**Figure 1**) composed of seven nodes, two of which (A and B) are designated as input nodes whose values remain constant during the update, and five regulated nodes ($C - G$) whose values are determined by the Boolean equations of the other nodes:

- $f_A = A,$
- $f_B = B,$
- $f_C = A \wedge \neg B,$
- $f_D = B \vee C,$
- $f_E = \neg A \wedge (C \vee D),$
- $f_F = D \wedge \neg C,$
- $f_G = (E \wedge F) \vee A.$

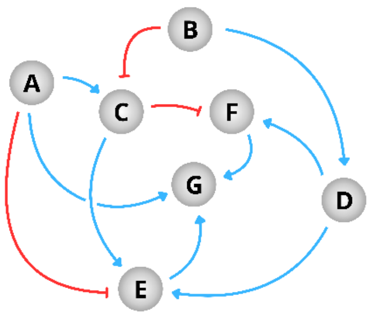


Figure 1. Schematic of the toy Boolean model composed of seven nodes (A-G). Nodes A and B are input nodes. Blue arrows represent activating interactions, and red arrows represent inhibitory interactions.

1. c(N)DNFs

Below we list the canonical disjunctive normal form ($cDNF(f_i)$) of each Boolean equation, alongside with the canonical disjunctive normal form of its negation ($cNDNF(f_i)$):

Function	$cDNF$	$cNDNF$
f_A	A	$\neg A$
f_B	B	$\neg B$
f_C	$A \wedge \neg B$	$\neg A \vee B$
f_D	$B \vee C$	$\neg B \wedge \neg C$
f_E	$(\neg A \wedge C) \vee (\neg A \wedge D)$	$A \vee (\neg C \wedge \neg D)$
f_F	$D \wedge \neg C$	$\neg D \vee C$
f_G	$A \vee (E \wedge F)$	$(\neg A \wedge \neg E) \vee (\neg A \wedge \neg F)$

2. Driver extraction and optimization by unit-lateral propagation

First, we will select s_2 (**Figure 2**) from the 4 different stable states reached by the toy model:

Stable state	A	B	C	D	E	F	G
s_1	0	0	0	0	0	0	0
s_2	**0**	**1**	**0**	**1**	**1**	**1**	**1**
s_3	1	0	1	1	0	0	1
s_4	1	1	0	1	0	1	1

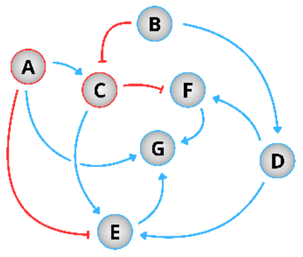


Figure 2. Visualization of the stable state s_2, selected as an example for driver extraction analysis. Nodes set to 1 (active) are highlighted in blue and nodes set to 0 (inactive) are highlighted in red.

For the stable state s_2, we extract the drivers (**Figure 3A**) as follows:

- If the node value is 1, select implicants from the $cDNF$ that evaluate to 1 in s_2 and then remove any negations, expressing each prime implicant as a conjunction of variables.
- If the node value is 0, select the true prime implicants from its $cNDNF$ in s_2 and similarly, drop the negations.

Next, we optimize each node's driver by collapsing trivial dependencies in an iterative fashion, where we (1) identify unitary drivers (those that have reduced to a single variable), (2) propagate unit drivers, in which every occurrence of each marked variable is then substituted into the driver formulas of its target nodes, replacing the intermediate node with its defining input, and (3) repeat to convergence until no driver remains a single literal.

Function	$cDNF$	$cNDNF$	s_2	$D(s_2)$	$D^*(s_2)$
f_A	A	$\neg A$	0	$\{A\}$	$\{A\}$
f_B	B	$\neg B$	1	$\{B\}$	$\{B\}$
f_C	$A \wedge \neg B$	$\neg A \vee B$	0	$\{A, B\}$	$\{A, B\}$
f_D	$B \vee C$	$\neg B \wedge \neg C$	1	$\{B\}$	$\{B\}$
f_E	$(\neg A \wedge C) \vee (\neg A \wedge D)$	$A \vee (\neg C \wedge \neg D)$	1	$\{A \wedge D\}$	$\{A \wedge B\}$
f_F	$D \wedge \neg C$	$\neg D \vee C$	1	$\{D \wedge C\}$	$\{B \wedge C\}$
f_G	$A \vee (E \wedge F)$	$(\neg A \wedge \neg E) \vee (\neg A \wedge \neg F)$	1	$\{E \wedge F\}$	$\{E \wedge F\}$

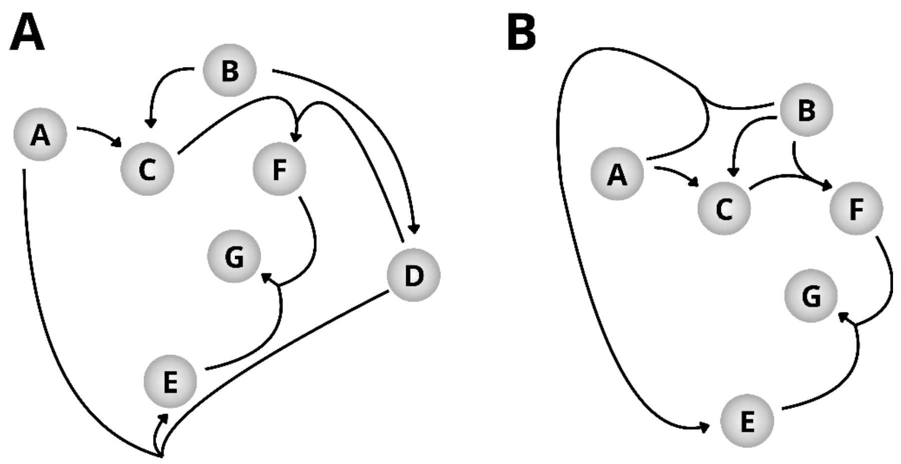


Figure 3. A. Drivers set graphical representation for s_2, obtaining by evaluating the cDNF (active) or cNDNF (inactive) of each node and selecting the prime implicants that are true, with negations removed. **B.** Graphical representation of the reduced drivers set for s_2, where variables whose drivers are unit-literal are replaced by the drivers of that unit-literal variable.

3. Path elongation, counting, and signed scoring

Now, we will count the paths toward G:

$$\boldsymbol{Paths_G} = \{(G)||p_E||p_F|(p_E, p_F) \in \boldsymbol{Paths_E} \times \boldsymbol{Paths_F}$$

1. Computing $Paths_E$ and $Paths_F$:

$$Paths_E = \{(E)||p_A||p_B|(p_A, p_B) \in Paths_A \times Paths_B\}$$
$$Paths_F = \{(F)||p_B||p_C|(p_B, p_C) \in Paths_B \times Paths_C\}$$

2. Computing $Paths_A$, $Paths_B$ and $Paths_C$:

$Paths_A = \{(A)\}$ (input)

$Paths_B = \{(B)\}$ (input)

$$Paths_C = \{(C)||p_A||p_B|(p_A, p_B) \in Paths_A \times Paths_B\} = \{(C)||(A)\} \cup \{(C)||(B)\} = \{(C, A), (C, B)\}$$

3. We got $Paths_A$, $Paths_B$ and $Paths_C$, now we substitute to get $Paths_E$ and $Paths_F$:

$$Paths_E = \{(E)||(A)||(B)\} = \{(E, A, B)\}$$

$$Paths_F = \{(F)||(B)||(C, A), (C, B)\} = \{(F, B, C, A), (F, B, C, B)\}$$

4. We can now get $Paths_G$:

$$\boldsymbol{Paths_G} = \{(G)||(E, A, B)||(F, B, C, A), (F, B, C, B)\} = \{(G, E, A, B, F, B, C, A), (G, E, A, B, F, B, C, B)\}$$

$$\boldsymbol{v_G(s_2)} = +|\{(G, E, A, B, F, B, C, A), (G, E, A, B, F, B, C, B)\}| = \mathbf{2\ paths}.$$

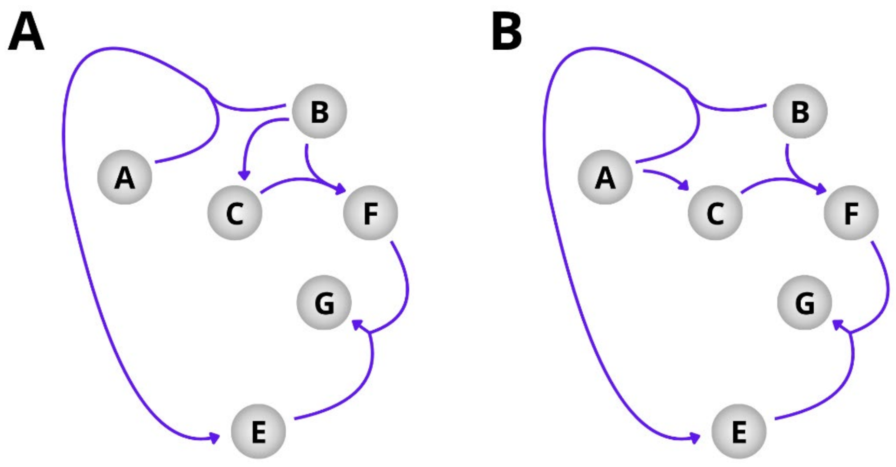


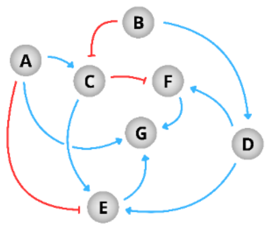
Figure 4. A-B. Graphical representation of the two paths towards G computed using BooLEVARD.

4. Topology-based path enumeration

In this section, we use a purely topological (network-based) approach to enumerate all paths leading to the activation of node G, and then compare the result to BooLEVARD's count.

Activating paths towards G

1. $G - F - \neg C - B$
2. $G - F - \neg C - \neg A$
3. $G - F - D - B$
4. $G - A$



5. $G - E - \neg A$
6. $G - E - D - B$

Hence, topologically, there are 6 possible activatory paths.

Corresponding inhibitory paths towards G

By flipping each element of the above paths, we obtain the six routes to inhibit G:

1. $\neg G - \neg F - C - \neg B$
2. $\neg G - \neg F - C - A$
3. $\neg G - \neg F - \neg D - \neg B$
4. $\neg G - \neg A$
5. $\neg G - \neg E - A$
6. $\neg G - \neg E - \neg D - \neg B$

Example: state s_2

Let's evaluate which of the six activating paths are "on" in the stable state s_2. Recall that in s_2, the node values are:

$$A = 0, B = 1, C = 0, D = 1, E = 1, F = 1, G = 1$$

Path	Evaluation	Valid
$G - F - \neg C - B$	1-1-!0-1 = 1-1-1-1	Yes
$G - F - \neg C - \neg A$	1-1-!0-!0 = 1-1-1-1	Yes
$G - F - D - B$	1-1-1-1	Yes
$G - A$	1-0	No
$G - E - \neg A$	1-1-!0	Yes
$G - E - D - B$	1-1-1-1	Yes

Out of 6 possible paths, 5 are active in s_2.

BooLEVARD finds 2 activating paths for G in s_2, whereas the purely topological enumeration finds 5. The discrepancy arises because:

- Topological enumeration considers only edge-directions (causality) and activation/inhibition signs but ignores logical conjunctions.
- BooLEVARD, in contrast, faithfully implements each Boolean equation, including all AND- and OR-relationships, so it only counts a path if every required conjunction at every elongation step is simultaneously satisfied.

In other words, a topology-only approach will overestimate path counts whenever the true Boolean rule contains an AND term that must be satisfied by multiple upstream inputs.