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 Disjunctive Normal Functions (and the canonical Disjunctive Normal Functions of their negations), (2) extract the drivers at a chosen stable state by selecting only the prime implicants that evaluate to true , (3) 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. Symbols and conventions follow those introduced by Crama and Hammer, 2011 (1).

Notations

$B = \{0,1\}$

$X = (x_1, x_2, \dots, x_n), Y = (y_1, y_2, \dots, y_n)$: components of points in B^n.

$$x^\alpha = \begin{cases} x, & if\ \alpha = 1, \\ \bar{x}, & if\ \alpha = 0. \end{cases}$$

f: Boolean function.

$T(f)$: the set of true points of function f.

$F(f)$: the set of false points of function f.

ϕ: Boolean expression.

s: stable state.

K: number of stable states.

$D(X, s_k)$: set of drivers in stable state k.

P: prime implicant (i.e. minterm).

1. Canonical Disjunctive Normal Functions

A minterm, or prime implicant (P) on B^n is an elementary conjunction involving exactly n literals of the form $x_1^{y_1} \wedge \dots \wedge x_n^{y_n}$. Let f be a Boolean function on B^n, let $T(f)$ be the set of true points of f, and let $F(f)$ be its set of false points. The canonical Disjunctive Normal Function (cDNF) (Eq. (1)) is the minterm expression of f.

$$\phi_f(X) = \bigvee_{Y \in T(f)} \left(\bigwedge_{i \mid y_i = 1} x_i \wedge \bigwedge_{j \mid y_j = 0} \overline{x_j} \right) \qquad (1)$$

1. Crama, Y., & Hammer, P. L. (2011). Boolean Functions: Theory, Algorithms, and Applications. Cambridge University Press.

Respectively, the canonical Disjunctive Normal Function of $\bar{f}$ (cNDNF) (Eq. (2)), is the disjunction of minterms corresponding to the false points of f.

$$\phi_{\bar{f}}(X) = \bigvee_{Y \in F(f)} \left(\bigwedge_{i \mid y_i = 1} x_i \wedge \bigwedge_{j \mid y_j = 0} \bar{x_j} \right) \tag{2}$$

2. Driver extraction and optimization by unit-lateral propagation

If a model has K stable states, given the stable state $s_k = \{0,1\}^n$, we extract for each node the driver $D(X, s_k)$ by selecting only those implicants that evaluate to true under s, then discard literal polarity, as elucidated in Eq. (3):

$$D(X, s_k) = \begin{cases} \{Var(P) \mid P \in \phi_{f(X)}\}, & s_k(X) = 1, \\ \{Var(P) \mid P \in \phi_{\bar{f}(X)}\}, & s_k(X) = 0. \end{cases} \tag{3}$$

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

$$\begin{gathered} \widehat{D}^{m+1}(X, s_k) = \widehat{D}^{m}(X, s_k)\left[Y \mapsto \widehat{D}^{m}(Y, s_k)\right]_{Y \in R^{(m)}} \\ R^m = \{Y \mid \exists\, \alpha \in \{0,1\}\colon \widehat{D}^{m}(Y, s_k) = \{x_Y^{\alpha}\}\} \end{gathered} \tag{4}$$

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, yielding the simplified drivers set $D^*(X, s_k)$.

3. Path elongation, counting, and signed scoring

From the fully optimized driver $D^*(X, s_k)$, 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 $Input$):

$$\begin{gathered} S_{Input} = \{x_i \in X \mid No\ input\ edge\ for\ x_i\}, \\ Paths_i = \begin{cases} \{\{i\}\}, & i \in S_{Input}, \\ \bigcup_{d = \{j_1, \dots, j_n\} \in D_i^*} \{(i) \parallel p_{j_1} \parallel \cdots \parallel p_{j_n} \mid (p_{j_1}, \dots, p_{j_n}) \in Paths_{j_k}, i \notin p_{j_k} \forall k\}, & i \notin S_{Input}. \end{cases} \end{gathered} \tag{5}$$

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

Next, in Eq. (6-7) we define the path count and attach a sign according to the node's state:

$$CP_i(s_k) = |Paths_i(s_k)|, k \in \{1, \dots, K\} \tag{6}$$

$$v_i(s_k) = \begin{cases} +CP_i(s), & s_k(x_i) = 1, \\ -CP_i(s), & s_k(x_i) = 0. \end{cases} \tag{7}$$

Finally, averaging over all K distinct stable states in Eq. (8) yields the model-level score vector (available through the `collapsed=True` argument within the `CountPaths` function).

$$V_i = \frac{1}{K}\sum_{k=1}^{K} v_i(s_k) \tag{8}$$

Toy example

We consider a Boolean model (**Figure 1**) composed of $n = 7$ 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:

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

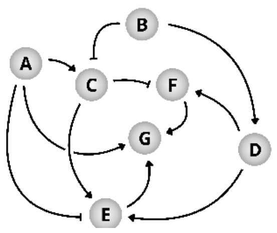


Figure 1. Schematic of the toy Boolean model composed of seven nodes (A-G). Nodes A and B are input nodes.

1. Canonical Disjunctive Normal Functions

Below we list the canonical Disjunctive Normal Form (ϕ_f) of each Boolean equation, alongside the canonical disjunctive normal form of its negation ($\phi_{\bar{f}}$):

Function	$\boldsymbol{\phi_f}$	$\boldsymbol{\phi_{\bar{f}}}$
$\boldsymbol{f_A}$	A	$\bar{A}$
$\boldsymbol{f_B}$	B	$\bar{B}$
$\boldsymbol{f_C}$	$A \wedge \bar{B}$	$\bar{A} \vee B$
$\boldsymbol{f_D}$	$B \vee C$	$\bar{B} \wedge \bar{C}$
$\boldsymbol{f_E}$	$(\bar{A} \wedge C) \vee (\bar{A} \wedge D)$	$A \vee (\bar{C} \wedge \bar{D})$
$\boldsymbol{f_F}$	$D \wedge \bar{C}$	$\bar{D} \vee C$
$\boldsymbol{f_G}$	$A \vee (E \wedge F)$	$(\bar{A} \wedge \bar{E}) \vee (\bar{A} \wedge \bar{F})$

2. Driver extraction and optimization by unit-lateral propagation

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

Stable state	***A***	***B***	***C***	***D***	***E***	***F***	***G***
$\boldsymbol{s_1}$	0	0	0	0	0	0	0
$\boldsymbol{s_2}$	**0**	**1**	**0**	**1**	**1**	**1**	**1**
$\boldsymbol{s_3}$	1	0	1	1	0	0	1
$\boldsymbol{s_4}$	1	1	0	1	0	1	1

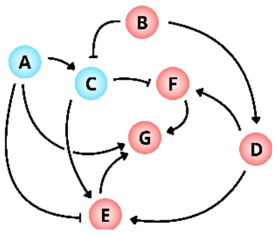


Figure 2. Visualization of the stable state s_2, selected as an example for driver extraction analysis. Nodes set to 1 (active) are highlighted in blue and nodes set to 0 (inactive) are highlighted in red.

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

- If the node value is 1, select implicants from the ϕ_f 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 $\phi_{\bar{f}}$ 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	$\boldsymbol{\phi_f}$	$\boldsymbol{\phi_{\bar{f}}}$	$\boldsymbol{s_2}$	$\boldsymbol{D(s_2)}$	$\boldsymbol{D^*(s_2)}$
$\boldsymbol{f_A}$	A	$\boldsymbol{\bar{A}}$	0	$\{\{A\}\}$	$\{\{A\}\}$
$\boldsymbol{f_B}$	$\boldsymbol{B}$	$\bar{B}$	1	$\{\{B\}\}$	$\{\{B\}\}$
$\boldsymbol{f_C}$	$A \wedge \bar{B}$	$\boldsymbol{\bar{A}} \vee \boldsymbol{B}$	0	$\{\{A\},\{B\}\}$	$\{\{A\},\{B\}\}$
$\boldsymbol{f_D}$	$\boldsymbol{B} \vee \boldsymbol{C}$	$\bar{B} \wedge \bar{C}$	1	$\{\{B\}\}$	$\{\{B\}\}$
$\boldsymbol{f_E}$	$(\boldsymbol{\bar{A}} \wedge \boldsymbol{C}) \vee (\boldsymbol{\bar{A}} \wedge \boldsymbol{D})$	$A \vee (\bar{C} \wedge \bar{D})$	1	$\{\{A, D\}\}$	$\{\{\boldsymbol{A}, \boldsymbol{B}\}\}$
$\boldsymbol{f_F}$	$\boldsymbol{D} \wedge \boldsymbol{\bar{C}}$	$\bar{D} \vee C$	1	$\{\{D, C\}\}$	$\{\{\boldsymbol{B}, \boldsymbol{C}\}\}$
$\boldsymbol{f_G}$	$A \vee (E \wedge F)$	$(\bar{A} \wedge \bar{E}) \vee (\bar{A} \wedge \bar{F})$	1	$\{\{E, F\}\}$	$\{\{E, F\}\}$

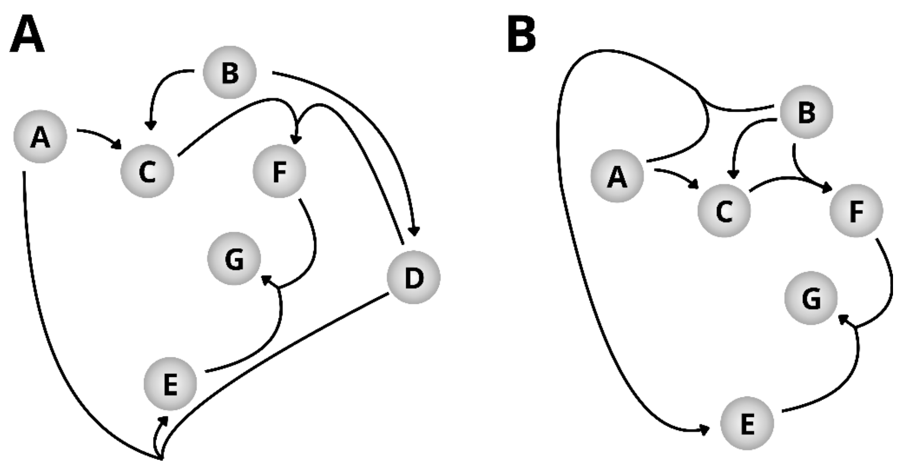


Figure 3. A. Drivers set graphical representation for s_2, obtaining by evaluating the cDNF (active) or cNDNF (inactive) of each node and selecting the prime implicants that are true, with negations removed. **B.** Graphical representation of the reduced drivers set for s_2, where variables whose drivers 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. Given that $D_G^* = \{E, F\}$:

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

1. Computing $Paths_E$ and $Paths_F$. Given that $D_E^* = \{A, B\}$ and that $D_F^* = \{B, C\}$:

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

2. Computing $Paths_A$, $Paths_B$ and $Paths_C$. Given that $D_A^* = \{A\}$, $D_B^* = \{B\}$, and $D_C^* = \{\{A\}, \{B\}\}$:
$Paths_A = \{(A)\}$ (input)

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

$Paths_C = \{(C) || p_A \mid p_A \in Paths_A, C \notin p_A\} \cup \{(C) || p_B \mid p_B \in Paths_B, C \notin p_B\}$

$Paths_C = \{(C)||(A)\} \cup \{(C)||(B)\} = \{(C, A), (C, B)\}$

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

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

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

Notice that in the second path for F, (F, B, C, B), the node B appears twice—but this is not evidence of a feedback loop. The first B comes from F's driver $\{B, C\}$, while the second B comes from the C's driver $\{\{A\}, \{B\}\}$ (**Figure 4**).

4. We can now get $Paths_G$:

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

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

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

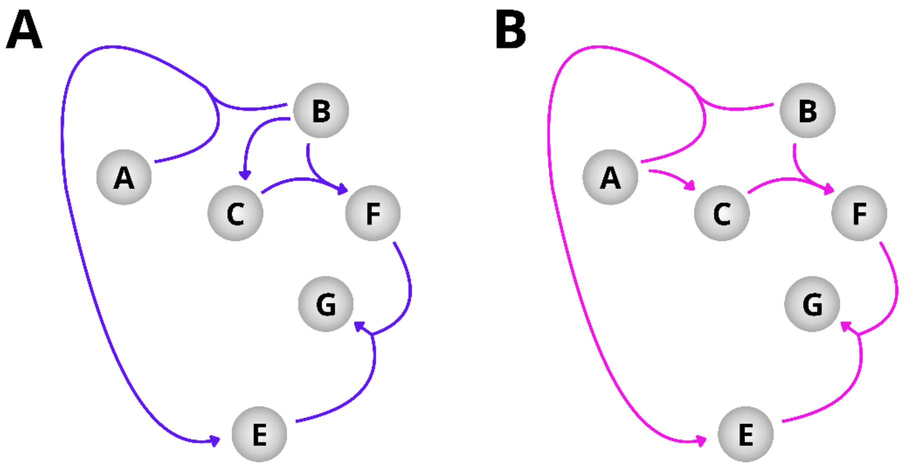


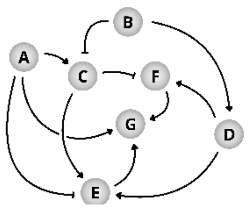
Figure 4. A-B. Graphical representation of the two paths towards G computed using BooLEVARD.

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. $G - F - \bar{C} - B$
2. $G - F - \bar{C} - \bar{A}$
3. $G - F - D - B$
4. $G - A$
5. $G - E - \bar{A}$
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. $\bar{G} - \bar{F} - C - \bar{B}$
2. $\bar{G} - \bar{F} - C - A$
3. $\bar{G} - \bar{F} - \bar{D} - \bar{B}$
4. $\bar{G} - \bar{A}$
5. $\bar{G} - \bar{E} - A$
6. $\bar{G} - \bar{E} - \bar{D} - \bar{B}$

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
$G - F - \bar{C} - B$	1-1-!0-1 = 1-1-1-1	Yes
$G - F - \bar{C} - \bar{A}$	1-1-!0-!0 = 1-1-1-1	Yes
$G - F - D - B$	1-1-1-1	Yes
$G - A$	1-0	No
$G - E - \bar{A}$	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 differences arise 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.