

Trap spaces of Boolean networks are conflict-free siphons of their Petri net encoding

Van-Giang Trinh^a, Belaid Benhamou^a, Sylvain Soliman^{b,*}

^a*LIS, Aix-Marseille University, Marseille, France*

^b*Lifeware team, Inria Saclay center, Palaiseau, France*

Abstract

Boolean network modeling of gene regulation but also of post-transcriptomic systems has proven over the years that it can bring powerful analyses and corresponding insight to the many cases where precise biological data is not sufficiently available to build a detailed quantitative model. Besides simulation, the analysis of such models is mostly based on attractor computation, since those correspond roughly to observable biological *phenotypes*. The recent use of trap spaces made a real breakthrough in that field allowing to consider medium-sized models that used to be out of reach. However, with the continuing increase in model size and complexity of Boolean update functions, the state-of-the-art computation of minimal trap spaces based on *prime-implicants* shows its limits due to the difficulty of the prime-implicant computation.

In this article we explore and prove for the first time a connection between trap spaces of a general Boolean network and siphons of its Petri net encoding. Besides important theoretical applications in studying properties of trap spaces, the connection enables us to propose an alternative approach to compute minimal trap spaces, and hence complex attractors, of a general Boolean network. It replaces the need for prime-implicants by a completely different technique, namely the enumeration of maximal siphons in the Petri net encoding of the original model. We then demonstrate its efficiency and compare it to the state-of-the-art methods on a large collection of real-world

*Corresponding author.

Email addresses: `trinh.van-giang@lis-lab.fr` (Van-Giang Trinh),
`belaid.benhamou@lis-lab.fr` (Belaid Benhamou), `Sylvain.Soliman@inria.fr`
(Sylvain Soliman)

and randomly generated models.

Keywords:

Logical model, Boolean network, Trap space, Attractor computation, Petri net, Siphon, Systems biology

1. Introduction

From the observation that the transcriptional regulation behaved in a sigmoid step-like way, came the original idea to represent models of gene regulation as discrete event systems. Those Gene Regulation Networks (GRN) use thresholds or equivalently logical functions to represent the different regulations [1, 2, 3, 4].

Boolean net modeling has proven over the years that it can bring powerful analyses and corresponding insight to the many cases where precise biological data is not sufficiently available to build a detailed quantitative model [5], even for modelling post-transcriptional mechanisms. This is even more true for very large models where such data is frequently missing and led to a constant increase in size of logical models *à la* Thomas [6]. Besides simulation, the analysis of such models is mostly based on attractor computation, since those correspond roughly to observable biological *phenotypes*. The recent use of trap spaces [7] made a real breakthrough in that field allowing to consider medium-sized models that used to be out of reach. However, with the continuing increase in model size and complexity of Boolean update functions, the state-of-the-art computation of minimal trap spaces based on *prime-implicants* shows its limits. More specifically, the number of prime implicants of a Boolean function is in general exponential in the number of input nodes of this function [7]. Moreover, the computation of prime implicants is a demanding task, especially for complex Boolean functions.

It is worth noting that the recent method presented in [8] for computing minimal trap spaces avoids the prime-implicants computation by relying on the *most-permissive* semantics of Boolean networks. This method has been implemented in the tool `mpbn`¹ demonstrated in [9] for handling medium-sized models from the literature and very large synthetic models (up to 100,000 nodes). However, this method is only applicable for *locally-monotonic* Boolean networks, whereas the prime-implicants based method [7]

¹<https://github.com/bnediction/mpbn>

30 is applicable for *general* Boolean networks (i.e., including both locally-monotonic
 31 and non-locally-monotonic ones). In addition, the `bioLQM` platform also pro-
 32 vides another method using Binary Decision Diagrams (BDDs) in [http://](http://colomoto.org/biolqm/doc/tools-trapspaces.html)
 33 colomoto.org/biolqm/doc/tools-trapspaces.html. This method avoids
 34 the prime-implicants computation as it characterizes the set of generic trap
 35 spaces of a Boolean network by a BDD, then filters this set to get the set
 36 of all minimal trap spaces. By this approach, it requires the computation
 37 of all solutions, whereas the ASP-based methods [7, 9] can start enumerat-
 38 ing them as they are found. Moreover, the main issue with the BDD-based
 39 method is that the number of generic trap spaces of a Boolean network may
 40 be extremely larger than the number of minimal trap spaces of this Boolean
 41 network. This issue limits the efficiency of the BDD-based method. The
 42 study [10] highlights the need for non-locally-monotonic Boolean networks
 43 in both biological and theoretical aspects. Hence, it is still necessary to
 44 develop efficient methods for computing minimal trap spaces of large-scale
 45 general Boolean networks.

46 Petri nets were introduced in the 60s as simple formalism for describing
 47 and analyzing information-processing systems that are characterized as be-
 48 ing concurrent, asynchronous, non-deterministic and possibly distributed [11,
 49 12]. The use of Petri nets for representing biochemical reaction systems, by
 50 mapping molecular species to places and reactions to transitions, hinted at
 51 already in [11, 12] was used more thoroughly quite late in [13], together with
 52 some Petri net concepts and tools for the analysis of metabolic networks.
 53 Siphons are such a concept, but they have not been used a lot for the study
 54 of biochemical systems [14, 15] even if the practical cost of computing their
 55 minimal/maximal elements appear much more manageable than the theoret-
 56 ical complexity would indicate [16, 17].

57 In this article we explore and prove for the first time a connection be-
 58 tween trap spaces of a general Boolean network and siphons of its Petri net
 59 encoding. Not only having important theoretical applications in studying
 60 properties of trap spaces in Boolean networks, the connection has impor-
 61 tant practical applications in the trap space computation. Specifically, based
 62 on the connection, we propose an alternative approach to compute minimal
 63 trap spaces, and hence complex attractors, of a general Boolean network. It
 64 replaces the need for prime-implicants by a completely different technique,
 65 namely the enumeration of maximal siphons in the Petri net encoding of the
 66 original model. We then demonstrate its efficiency and compare it to the
 67 state-of-the-art methods for computing minimal trap spaces in Boolean net-

works on many real-world models from various sources in the literature and randomly generated models.

Herein we revise and extend our previous work in [18] as follows. First, more formal definitions are given and the existing proofs are made more detailed. In particular, an updated proof provides another way to prove the independence of trap spaces of a Boolean network on its update scheme, which was originally proved in [7]. Second, we showcase a theoretical application of the connection between trap spaces in Boolean networks and conflict-free siphons in Petri nets. Third, beyond the proposed ASP method implementing the alternative approach [18], we propose several other possible methods for computing minimal trap spaces using Maximum Satisfiability (MaxSAT), Constraint Logic Programming (CLP), and Integer Linear Programming (ILP). Fourth, we discuss in detail how to compute several special types of trap spaces in a Boolean network. Fifth, we present the idea for using our Petri net approach to handle the problem of inconsistent and incomplete data. Sixth, regarding the implementation, we have developed a new converter that directly reads a `.bnet` file and builds the Petri net encoding, instead of using the PNML conversion of `bioLQM` [18]. Finally, we conduct a more comprehensive benchmark on more real-world models from various sources and randomly generated models to evaluate all the proposed methods (the benchmark conducted in [18] considers only real-world models).

The rest of this paper is organized as follows: Section 2 recalls the basic concepts including Boolean networks, attractors, trap spaces, Petri nets, and siphons. Section 3 presents the main finding, the connection between trap spaces in Boolean networks and siphons in Petri nets. Section 4 presents the alternative approach for computing minimal trap spaces and the four possible methods implementing it. Section 5 shows an important biological case study showing the applicability of the new approach. Section 6 reports the experimental results for evaluating the efficiency of the proposed methods. Finally, Section 7 concludes the paper and draws future work.

2. Preliminaries

We shall briefly recall here some preliminaries on Boolean networks related to trap spaces and Petri nets. **Remove this statement because there is not sure if the encoded Boolean network preserves the trap spaces of the original multi-level logical model.**

103 2.1. Boolean networks

104 **Definition 2.1.** A Boolean Network (BN) is a pair $\mathcal{N} = (V, F)$ where:

- 105 • $V = \{v_1, \dots, v_n\}$ is the set of nodes. We use v_i to denote both the node
106 v_i and its associated Boolean variable.
- 107 • $F = \{f_1, \dots, f_n\}$ is the set of update functions. Each function f_i is
108 associated with node v_i and satisfies $f_i: \mathbb{B}^{|IN(v_i)|} \mapsto \mathbb{B}$ where $\mathbb{B} = \{0, 1\}$
109 and $IN(v_i)$ denotes the set of input nodes of v_i . Note that a node $v_i \in V$
110 is called a source node if and only if $f_i = v_i$.

111 A Boolean function is *locally-monotonic* if it can be represented by a
112 formula in disjunctive normal form in which all occurrences of any given
113 literal are either negated or non-negated [9]. A Boolean network is said
114 to be locally-monotonic if all its Boolean functions are locally-monotonic.
115 Otherwise, this model is said to be non-locally-monotonic.

116 A state $v \in \mathbb{B}^n$ is as a mapping $v: V \mapsto \mathbb{B}$ that assigns either 0 (inactive)
117 or 1 (active) to each node. We denote the set of all possible states of a
118 Boolean network \mathcal{N} by $\mathcal{S}_{\mathcal{N}} = \mathbb{B}^n$. At each time step t , node v_i can update
119 its state by

$$v_i(t+1) = f_i(v(t))$$

120 where $v(t)$ is the state of \mathcal{N} at time t and $v_i(t+1)$ is the state of node v_i at
121 time $t+1$. Note that for simplicity, we write $f_i(v(t))$ even $IN(v_i) \subset V$ (i.e.,
122 $IN(v_i)$ does not contain some nodes of V). An update scheme of a Boolean
123 network specifies the way that the nodes update their states through time
124 evolution [4]. Following the update scheme, the Boolean network transits
125 from a state to another state (possibly identical). This transition is called
126 the *state transition* and denoted by $\rightarrow \subseteq \mathcal{S}_{\mathcal{N}} \times \mathcal{S}_{\mathcal{N}}$. Then the dynamics of \mathcal{N}
127 is captured by the directed graph $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ called the State Transition Graph
128 (STG). There are two main types of update schemes [4]: synchronous, where
129 all the nodes are update simultaneously, and fully asynchronous, where only
130 one node is nondeterministically selected to be updated.

131 2.2. Traps spaces

132 We recall here some definitions from [7] for the introduction of *trap spaces*.
133 Minimal trap spaces prove to be a very good approximation of the attractors
134 of a Boolean network under asynchronous update schemes and have become
135 the *de facto* standard way to analyze models of a few tens of *genes* [19, 20].

136 An non-empty set $T \subseteq \mathcal{S}_{\mathcal{N}}$ is a trap set with respect to \rightarrow if for every
 137 $x \in T$ and $y \in S$ with $x \rightarrow y$ it holds that $y \in T$ [7]. An attractor of \mathcal{N}
 138 with respect to \rightarrow can be defined as an inclusion-wise minimal trap set of
 139 $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$. An attractor can be also seen as a terminal strongly connected
 140 component of $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ [21]. An attractor of size 1 is called a fixed point,
 141 otherwise a cyclic attractor [7].

142 A subspace m of a Boolean network $\mathcal{N} = (V, F)$ is a mapping $m: V \mapsto$
 143 $\mathbb{B} \cup \{\star\}$. $m(v_i) \in \mathbb{B}$ means that the value of v_i is fixed in m and v_i is called
 144 a fixed variable. $m(v_i) \in \star$ means that the value of v_i is free in m and v_i is
 145 called a free variable. We denote D_m the set of all fixed variables of m . A
 146 subspace m is equivalent to a set of states:

$$\mathcal{S}_{\mathcal{N}}[m] := \{s \in \mathcal{S}_{\mathcal{N}} \mid \forall v \in D_m: s(v) = m(v)\}.$$

147 For example, $m = \star \star 1$ (for simplicity, we write subspaces likes states)
 148 means that $D_m = \{v_3\}$, $m(v_3) = 1$, and it is equivalent to the set of states
 149 $\{001, 011, 101, 111\}$. We denote $\mathcal{S}_{\mathcal{N}}^* = (\mathbb{B} \cup \{\star\})^n$ the set of all possible
 150 subspaces of \mathcal{N} . Note that $|\mathcal{S}_{\mathcal{N}}^*| = 3^n$ and $\mathcal{S}_{\mathcal{N}} \subset \mathcal{S}_{\mathcal{N}}^*$ [7].

151 A *trap space* is defined as a subspace that is also a trap set. It is noted
 152 that trap spaces of a Boolean network are independent of the update scheme
 153 of this model [7]. Then, we define a partial order $<$ on $\mathcal{S}_{\mathcal{N}}^*$ as: $m < m'$ if and
 154 only if $\mathcal{S}_{\mathcal{N}}[m] \subseteq \mathcal{S}_{\mathcal{N}}[m']$ and $\mathcal{S}_{\mathcal{N}}[m] \neq \mathcal{S}_{\mathcal{N}}[m']$. Consequently, a trap space m
 155 is minimal if and only if there is no trap space $m' \in \mathcal{S}_{\mathcal{N}}^*$ such that $m' < m$.

156 For example, let us consider the Boolean network shown in Example 2.1.
 157 Figure 1(a) shows the dynamics of this model under the fully asynchronous
 158 update (i.e., only one node is nondeterministically selected in order to be
 159 updated at each time step). The model has all two trap spaces, $m_1 = 11$
 160 and $m_2 = \star\star$. Since $m_1 < m_2$, m_1 is a minimal trap space of the Boolean
 161 network.

162 **Example 2.1.** We give a Boolean network $\mathcal{N} = (V, F)$, where $V = (x_1, x_2)$
 163 and $F = (f_1, f_2)$ with $f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$, $f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$.
 164 Herein, \wedge , \vee , and \neg denote the conjunction, disjunction, and negation logical
 165 operators, respectively.

166 2.3. Petri net encoding of Boolean networks

167 **Definition 2.2.** A Petri net is a weighted bipartite directed graph (P, T, W) ,
 168 where P is a non-empty finite set of vertices called places, T is a non-empty



Figure 1: Dynamics and encoding of the Boolean network of Example 2.1

169 *finite set of vertices called transitions, $P \cap T = \emptyset$, and $W : (P \times T) \cup (T \times P) \mapsto$*
 170 *\mathbb{N} is a weight function attached to the arcs.*

171 A *marking* for a Petri net is a mapping $m : P \mapsto \mathbb{N}$ that assigns a number
 172 of tokens to each place. A place p is marked by a marking m if and only if
 173 $m(p) > 0$. Marking m can be seen as a subset of P that contains all marked
 174 places by m . We shall write $\text{pred}(x)$ (resp. $\text{succ}(x)$) to represent the set of
 175 vertices that have a (non-zero weighted) arc leading to (resp. coming from) x .
 176 In this work, we consider a class of Petri nets called 1-safe Petri nets where
 177 every place has at most 1 token and all arcs are of weight 1. In this case,
 178 weights are implicitly omitted in the arcs of a Petri net. Then, a transition
 179 $t \in T$ is *enabled* at a marking m if and only if $\text{pred}(t) \subseteq m$. A marking m
 180 is called a *deadlock* if there are no enabled transitions at m . The firing of
 181 t leads to a new marking m' specified by $m' = (m \setminus \text{pred}(t)) \cup \text{succ}(t)$. Note
 182 that when multiple transitions are enabled, we need to embed one firing
 183 scheme (similar to the update scheme of a Boolean network) to the Petri
 184 net. The classical firing scheme is that only one of the enabled transition is
 185 non-deterministically chosen to fire [12].

186 The link between Boolean networks *à la* Thomas and Petri nets was
 187 originally established in [22] in order to make available formal methods like
 188 model-checking for the analysis of such systems. The basic encoding into 1-
 189 safe (i.e., never more than one token in each place) nets only holds for purely
 190 Boolean networks but was later extended to multivalued logical models in
 191 two ways, either in [23] with non 1-safe Petri nets or more recently in [21]
 192 with 1-safe nets but many more places.

193 Since our study is focused on Boolean networks, we briefly recall the origi-
 194 nal encoding here. Its basis is that every node (*gene*) v of the original model
 195 $\mathcal{N} = (V, F)$ is represented by two separate places (p_v and \bar{p}_v), corresponding

196 to its two states, active, and inactive, respectively. Each conjunct of the
 197 logical function that activates the *gene* will lead to a transition t , consuming
 198 the inactive place (i.e., a directional arc from \bar{p}_v to t), producing the active
 199 place (i.e., a directional arc from t to p_v), and with all other literals both
 200 consumed and produced (i.e., a bidirectional arc). And conversely for the
 201 inactivation. Let s be a state of the Boolean network and m_s be its corre-
 202 sponding marking in the encoded Petri net. It holds that $\forall v \in V, s(v) = 0$ if
 203 and only if $m_s(\bar{p}_v) = 1$ and $s(v) = 1$ if and only if $m_s(p_v) = 1$. Note also that
 204 at any marking m of the Petri net encoding a Boolean network, it always
 205 holds that $m(p_v) + m(\bar{p}_v) = 1$.

206 The main property of this encoding is that it is completely faithful with
 207 respect to the update scheme of the original Boolean network. For each node
 208 v of \mathcal{N} , only transitions corresponding to v can change the current marking
 209 of p_v or \bar{p}_v . In addition, at any marking at most one of such transitions is en-
 210 abled because $m(p_v) + m(\bar{p}_v) = 1$ holds. Hence, for any update scheme in \mathcal{N} ,
 211 we have a corresponding firing scheme in \mathcal{P} , which preserves the equivalence
 212 between the dynamics of \mathcal{N} and \mathcal{P} [24].

213 For illustration, let us reconsider the Boolean network shown in Exam-
 214 ple 2.1. Figure 1(b) shows the Petri net encoding of this Boolean network.
 215 Place p_{x_1} (resp. \bar{p}_{x_1}) in \mathcal{P} represents the activation (resp. the inactivation) of
 216 node x_1 in \mathcal{N} . Marking $\{p_{x_1}, \bar{p}_{x_2}\}$ in \mathcal{P} represents state 10 in \mathcal{N} . Transitions
 217 $t_{x_1}^1$ and $t_{x_1}^2$ represent the update of node x_1 . Of course, in any marking $t_{x_1}^1$
 218 and $t_{x_1}^2$ cannot be both enabled. Then, the fully asynchronous update scheme
 219 in \mathcal{N} corresponds to the classical firing scheme in \mathcal{P} where only one of the
 220 enabled transitions for a given marking will be fired [12].

221 Note that given a Boolean network in the standard SBML-Qual for-
 222 mat [25], i.e., the package of SBML v3 [26] for such models, one can easily
 223 obtain its Petri net encoding in the Petri Net Markup Language (PNML)²
 224 standard using the `bioLQM`³ library. This piece of software extracted from
 225 GINsim [27] and part of the CoLoMoTo⁴ [28] software suite allows for easy
 226 conversion between standard formats. It also accepts many other common
 227 formats for Boolean networks, notably the `.bnet` files of the BoolNet [29, 19]
 228 tools. The conversion is executed as follows:

²<https://www.pnml.org/>

³<http://www.colomoto.org/biolqm/>

⁴<http://colomoto.org/>

229 `java -jar GINsim.jar -lqm <input.{sbml,bnet,zginml,...}> <output.pnml>`

230 Note that transforming a Boolean network defined by its functions into its
 231 Petri net encoding roughly relies on obtaining conditions for the activation
 232 and inactivation of the states. In [22] this took the form of the whole truth
 233 table of the Boolean functions, but as shown in Appendix 1 of [21] comput-
 234 ing Disjunctive Normal Forms (DNF) of each Boolean function is enough.
 235 Though this might appear quite computationally intensive it is important to
 236 remark first that contrary to the prime-implicants case, there is no need to
 237 find *minimal* DNFs. One way to look at this is to consider that this amounts
 238 to a similar approach as that used in [8] but with the encoding of both activa-
 239 tion and inhibition functions as DNFs in order to take into account possible
 240 non-local-monotonicity. This does not change the worst-case-complexity (ob-
 241 taining a single DNF being exponential) but might matter a lot in practice.
 242 As such, we will explore how this transformation, here using BDDs in `bioLQM`,
 243 and the one based on the most-permissive semantics compare in the Section 6
 244 on evaluation.

245 2.4. Siphons

246 Siphons are a static and classical property of Petri nets [11]. Note how-
 247 ever that the use of siphons for the analysis of biological models, though it is
 248 not new, has been mostly relevant to the ODE-based continuous semantics
 249 of Chemical Reaction Networks [30, 31, 32]. We recall here the basic defini-
 250 tion establishing that to produce something in a siphon you must consume
 251 something from the siphon. This corresponds to the idea that a siphon is a
 252 set of places that once unmarked remains unmarked.

253 **Definition 2.3.** *A siphon of a Petri net (P, T, W) is a set of places S such*
 254 *that:*

$$\forall t \in T, S \cap \text{succ}(t) \neq \emptyset \Rightarrow S \cap \text{pred}(t) \neq \emptyset.$$

255 *Note that \emptyset is trivially a siphon.*

256 Let $\text{pred}(S) := \bigcup_{s \in S} \text{pred}(s)$ and $\text{succ}(S) := \bigcup_{s \in S} \text{succ}(s)$. If $S = \emptyset$, then
 257 conventionally $\text{pred}(S) = \text{succ}(S) = \emptyset$. We have an important property on
 258 siphons [33] as follows.

259 **Proposition 2.1.** *Let S be a siphon of a Petri net (P, T, W) . Then $\text{pred}(S) \subseteq$*
 260 *$\text{succ}(S)$.*

261 3. Minimal trap spaces as maximal conflict-free siphons

262 First, we add a definition related to any set of places of a Petri net
263 encoding a Boolean network, and notably a siphon of such a net.

264 **Definition 3.1.** *A set of places of Petri net \mathcal{P} encoding Boolean network*
265 *\mathcal{N} is conflict-free if it does not contain any two places corresponding to the*
266 *active and inactive states of the same node of \mathcal{N} . Then, a conflict-free siphon*
267 *S is said to be maximal if and only if there is no other conflict-free siphon*
268 *S' such that $S \subset S'$.*

269 Intuitively, a siphon is a set of places that once unmarked remains so.
270 If it is conflict-free then its dual corresponds to a partial-state of the model
271 such that whatever update, the fixed values remain so (since the unmarked
272 places remain unmarked). This is precisely the definition of a trap space and
273 maximality of the siphon is equivalent to as many fixed values as possible,
274 hence minimality of the trap space. For example, the Boolean network given
275 in Example 2.1 has two trap spaces, $m_1 = 11$ and $m_2 = \star\star$. The Petri net
276 encoding of this Boolean network has five generic siphons, $S_1 = \emptyset$, $S_2 =$
277 $\{p_{x_1}, \bar{p}_{x_1}\}$, $S_3 = \{p_{x_2}, \bar{p}_{x_2}\}$, $S_4 = \{\bar{p}_{x_1}, \bar{p}_{x_2}\}$, and $S_5 = \{p_{x_1}, \bar{p}_{x_1}, p_{x_2}, \bar{p}_{x_2}\}$.
278 However, only S_1 and S_4 are conflict-free siphons and correspond to m_2 and
279 m_1 , respectively. Since $S_1 \subset S_4$, S_4 is a maximal siphon corresponding to
280 the minimal trap space m_1 . Hereafter, we formally prove that a (maximal)
281 conflict-free siphon is equivalent to a (minimal) trap space.

282 **Definition 3.2.** *Let m be a subspace of Boolean network $\mathcal{N} = (V, F)$. A*
283 *mirror of m is a set of places S in the Petri net encoding \mathcal{P} of \mathcal{N} such that:*

$$\forall v \in D_m, m(v) = 0 \Leftrightarrow p_v \in S, m(v) = 1 \Leftrightarrow \bar{p}_v \in S$$

284 and

$$\forall v \in V \setminus D_m, p_v \notin S, \bar{p}_v \notin S.$$

285 **Theorem 3.1.** *Let $\mathcal{N} = (V, F)$ be a Boolean network and \mathcal{P} be its Petri net*
286 *encoding. A subspace m is a trap space of \mathcal{N} if and only if its mirror S is a*
287 *conflict-free siphon of \mathcal{P} .*

288 *Proof.* First, we show that if m is a trap space of \mathcal{N} , then S is a conflict-free
289 siphon of \mathcal{P} (*). If $D_m = \emptyset$, then $S = \emptyset$ is trivially a conflict-free siphon of
290 \mathcal{P} . Thus, we consider the case that $D_m \neq \emptyset$ (resp. $S \neq \emptyset$). Assume that S is

291 not a siphon of \mathcal{P} . Then, there is a transition $t \in T$ such that $S \cap \text{succ}(t) \neq \emptyset$
 292 but $S \cap \text{pred}(t) = \emptyset$. This implies that there is a place $p \in S$ such that
 293 $p \in \text{succ}(t)$ but $p \notin \text{pred}(t)$. Let v be the corresponding node in \mathcal{N} of p . By
 294 the characteristics of the encoding [22], there is a directional arc from t to p
 295 and a directional arc from the complementary place of p to t . Without loss
 296 of generality, we assume that $p = p_v$, then there is a directional arc from t
 297 to p_v and a directional arc from \bar{p}_v to t . We follow the following procedure
 298 to find a state $s \in \mathcal{S}_{\mathcal{N}}[m]$ such that $m_s(p') = 1, \forall p' \in \text{pred}(t)$ where m_s is
 299 the corresponding marking in \mathcal{P} of s . For every place $p' \in \text{pred}(t)$, let p'' be
 300 the complementary place of p' and v' be the corresponding node in \mathcal{N} of p'
 301 and p'' . If $p'' \notin S$, then $v' \notin D_m$ and we can always set a Boolean value to
 302 $s(v')$ such that $s \in \mathcal{S}_{\mathcal{N}}[m]$ and $m_s(p') = 1$. If $p'' \in S$, then $v' \in D_m$ and we
 303 set $s(v') = m(v')$. In this case, if $p' = p_v$ then $s(v') = m(v') = 1$ leading to
 304 $m_s(p') = 1$, if $p' = \bar{p}_v$ then $s(v') = m(v') = 0$ leading to $m_s(p') = 1$. For
 305 the remaining nodes of \mathcal{N} , we can always set Boolean values to these nodes
 306 to preserve that $s \in \mathcal{S}_{\mathcal{N}}[m]$. We also have $m_s(p_v) = 0$ by the characteristics
 307 of the encoding [22]. Now, t is enabled at marking m_s . Its firing leads to
 308 a new marking m'_s such that $m'_s(p_v) = 1$ and $m'_s(\bar{p}_v) = 0$. Let s' be the
 309 corresponding state in \mathcal{N} of m'_s . We have $s'(v) = 1$ because $m'_s(p_v) = 1$ and
 310 $m(v) = 0$ because $p_v \in S$. This implies that $s' \notin \mathcal{S}_{\mathcal{N}}[m]$. For any firing
 311 scheme of \mathcal{P} , the firing of t always happens. Since a firing scheme of \mathcal{P} is
 312 equivalent to an update scheme of \mathcal{N} , s can escape from the trap space m
 313 for any update scheme of \mathcal{N} , which contradicts to the property of a trap
 314 space. Hence, S is a siphon of \mathcal{P} . By the definition of a mirror, S is also a
 315 conflict-free one.

316 Second, we show that if S is a conflict-free siphon of \mathcal{P} , then m is a trap
 317 space of \mathcal{N} (**). By the definition of a mirror, m is a subspace of \mathcal{N} . Let
 318 s be an arbitrary state in $\mathcal{S}_{\mathcal{N}}[m]$ and m_s be its corresponding marking in
 319 \mathcal{P} . Assume that there is a place $p \in S$ such that $m_s(p) = 1$. Let v be the
 320 corresponding node in \mathcal{N} of p . Since $p \in S$, $v \in D_m$ and $m(v) = s(v)$. If
 321 $p = p_v$, then $m_s(p_v) = 1$ leading to $m(v) = s(v) = 1$ by the characteristics of
 322 the encoding [22]. By the definition of a mirror, $m(v) = 0$ because $p_v \in S$,
 323 which is a contradiction. It is symmetric for the case that $p = \bar{p}_v$. Hence,
 324 $m_s(p) = 0, \forall p \in S$. In any marking m'_s reachable from m_s regardless of the
 325 firing scheme of \mathcal{P} , we have $m'_s(p) = 0, \forall p \in S$ by the dynamical property on
 326 markings of a siphon [33]. Let s' be the corresponding state in \mathcal{N} of m'_s . For
 327 every node $v \in D_m$, we have all two cases as follows. Case 1: $p_v \in S$, then
 328 $m'_s(p_v) = 0$, thus $s'(v) = 0 = m(v)$. Case 2: $\bar{p}_v \in S$, then $m'_s(\bar{p}_v) = 0$, thus

329 $s'(v) = 1 = m(v)$. Hence, $s'(v) = m(v)$ for every $v \in D_m$. Then, $s' \in \mathcal{S}_{\mathcal{N}}[m]$.
 330 By the definition of a trap space and the arbitrariness of s , m is a trap space
 331 of \mathcal{N} .

332 From (*) and (**), we can conclude the proof. □

333 From the proof of Theorem 3.1, we can see that this theorem still holds
 334 for any update scheme of the Boolean network. Since the Petri net encoding
 335 of a Boolean network is independent of its update scheme and siphons are
 336 a static property of a Petri net, we can imply that trap spaces of a Boolean
 337 network are independent of its update scheme. Note that the original proof
 338 for this property of trap spaces (see Theorem 1 of [7]) only considers the two
 339 popular update schemes (i.e., synchronous and fully asynchronous). This
 340 exhibits the very first theoretical application of the connection between trap
 341 spaces of Boolean networks and siphons of Petri nets.

342 **Theorem 3.2.** *Let \mathcal{N} be a Boolean network and \mathcal{P} be its Petri net encoding.*
 343 *A subspace m is a minimal trap space of \mathcal{N} if and only if its mirror S is a*
 344 *maximal conflict-free siphon of \mathcal{P} .*

345 *Proof.* First, we show that if m is a minimal trap space of \mathcal{N} , then S is
 346 a maximal conflict-free siphon of \mathcal{P} (*). Since m is a trap space of \mathcal{N} ,
 347 S is a conflict-free siphon of \mathcal{P} by Theorem 3.1. Assume that S is not
 348 maximal. Then, there is another conflict-free siphon S' such that $S \subset S'$.
 349 By Theorem 3.1, there is a trap space m' corresponding to S' . Following the
 350 definition of a mirror, $D_m \subset D_{m'}$ and $m(v) = m'(v), \forall v \in D_m$. It follows
 351 that $\mathcal{S}_{\mathcal{N}}[m'] \subset \mathcal{S}_{\mathcal{N}}[m]$, thus $m' < m$. This contradicts to the minimality of
 352 m . Hence, S is a maximal conflict-free siphon of \mathcal{P} .

353 Second, we show that if S is a maximal conflict-free siphon of \mathcal{P} , then
 354 m is a minimal trap space of \mathcal{N} (**). Since S is a conflict-free siphon of \mathcal{P} ,
 355 m is a trap space of \mathcal{N} by Theorem 3.1. Assume that m is not minimal.
 356 Then, there is another trap space m' such that $m' < m$. By the definition of
 357 the partial order $<$ on subspaces, $\mathcal{S}_{\mathcal{N}}[m'] \subset \mathcal{S}_{\mathcal{N}}[m]$. Let S' be the mirror of
 358 m' . S' is a conflict-free siphon by Theorem 3.1. Following the definition of
 359 a mirror, $S \subset S'$, which contradicts to the maximality of S . Hence, m is a
 360 minimal trap space of \mathcal{N} .

361 From (*) and (**), we can conclude the proof. □

362 We here showcase a theoretical application of the connection between trap
 363 spaces in Boolean networks and conflict-free siphons in Petri nets. We use it

364 to prove a property of minimal trap spaces, which has surprisingly not been
 365 formally proved. Specifically, all minimal trap spaces of a Boolean network
 366 are mutually disjoint. This property is important because we can use it to
 367 approximate the set of attractors of the Boolean network [7].

368 **Theorem 3.3.** *Let $\mathcal{N} = (V, F)$ be a Boolean network. For any two distinct*
 369 *minimal trap spaces m_1 and m_2 of \mathcal{N} , we have that $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$.*

370 *Proof.* Let \mathcal{P} be the Petri net encoding of \mathcal{N} . If \mathcal{N} has only one minimal
 371 trap space, then the theorem trivially holds. Note that by Theorem 3.2,
 372 \mathcal{N} always has at least one minimal trap space because \mathcal{P} has at least one
 373 maximal conflict-free siphon. Hence, we consider the case that \mathcal{N} has at least
 374 two minimal trap spaces.

375 Consider two any distinct minimal trap spaces m_1 and m_2 . Assume that
 376 $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] \neq \emptyset$. Let S_1 and S_2 be the mirrors of m_1 and m_2 , re-
 377 spectively. By Theorem 3.2, S_1 and S_2 are maximal conflict-free siphons
 378 of \mathcal{P} . We have that $S = S_1 \cup S_2$ is also a siphon because of Proposi-
 379 tion 2.1. For every node $v \in V$, assume that $p_v \in S$ and $\bar{p}_v \in S$ hold.
 380 Since S_1 and S_2 are conflict-free, there are all two cases. Case 1: $p_v \in S_1$
 381 and $\bar{p}_v \in S_2$. Case 2: $p_v \in S_2$ and $\bar{p}_v \in S_1$. These two cases lead to
 382 $m_1(v) \neq m_2(v), m_1(v) \neq \star, m_2(v) \neq \star$, then $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$. This is a
 383 contradiction. Hence, for every node $v \in V$, $p_v \in S$ and $\bar{p}_v \in S$ cannot hold
 384 together. Therefore, S is conflict-free. Now, we have that S is a conflict-free
 385 siphon but $S_1 \subset S$ or $S_2 \subset S$ holds because $S_1 \neq S_2$. This contradicts to the
 386 maximality of S_1 and S_2 . Hence, $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$ holds.

387 □

388 One naturally computational application of Theorem 3.1 is that we can ef-
 389 ficiently decide whether a subspace m is a trap space. In `PyBoolNet` [19], this
 390 is checked by using the percolation on the prime-implicants of the Boolean
 391 functions. As we have mentioned at the beginning of this article, the compu-
 392 tation of prime-implicants is a demanding task for complex Boolean networks,
 393 even is sometimes intractable. Hence, the checking method in [19] shows its
 394 limitations. Instead, we can first compute the mirror S_m of m in the Petri
 395 net encoding. Then, by Proposition 2.1 and Theorem 3.1, we can check if
 396 $\text{pred}(S_m) \subseteq \text{succ}(S_m)$. Note that the Petri net construction is less com-
 397 putationally demanding than the prime-implicant computation because it
 398 only requires computing generic (not prime) implicants of the Boolean func-

399 tions [21]. In addition, the time complexity of the above checking method is
400 quadratic in the number of transitions of the Petri net in worst cases.

401 Furthermore, by Theorem 3.2, we can reduce the problem of computing
402 all minimal trap spaces of a Boolean network to the problem of computing
403 all maximal conflict-free siphons of its Petri net encoding. Note that in the
404 case of special types of trap spaces (e.g., fixed points), this can be put in
405 regard to special types of siphons in Petri nets. See Subsection 4.5 for more
406 discussions about many special types of trap spaces. It might actually be
407 possible to generalize our result to any 1-safe place-complementary Petri net
408 to define a notion of trap spaces that might be useful for the analysis of Petri
409 nets, but this is out of the scope of this present article.

410 It is noted that there are no existing methods specifically designed for
411 computing maximal conflict-free siphons (even maximal siphons) of a Petri
412 net. The reason might be that researchers mainly focus on minimal generic
413 siphons [33] in the field of Petri nets. Hence, we here propose several methods
414 for computing maximal conflict-free siphons of a Petri net. The details of
415 the proposed methods shall be given in the next section.

416 4. Computation methods

417 4.1. Characterization

418 First, we show the characterization of all conflict-free siphons of the en-
419 coded Petri net $\mathcal{P} = (P, T, W)$. Suppose that S is a generic siphon of \mathcal{P} .
420 If a place p should belong to S , then by Proposition 2.1 all the transitions
421 in $\text{pred}(p)$ must belong to $\text{succ}(S)$. A transition t belongs to $\text{succ}(S)$ if and
422 only if there is at least one place p' in S such that $p' \in \text{pred}(t)$. Hence, for
423 each transition $t \in \text{pred}(p)$, we can state that

$$p \in S \Rightarrow \bigvee_{p' \in \text{pred}(t)} p' \in S. \quad (1)$$

424 The system of all the rules of the above form with respect to all pairs (p, t)
425 where $p \in P, t \in T, t \in \text{pred}(p)$ fully characterizes all generic siphons of a
426 Petri net and has been used with SAT solvers in [16, 17]. To make S to be
427 a conflict-free siphon, we need to add to the system the rule

$$p_v \in S \Rightarrow \bar{p}_v \notin S \wedge \bar{p}_v \in S \Rightarrow p_v \notin S \quad (2)$$

428 for each node $v \in V$. By definition, the final system fully characterizes all
429 conflict-free siphons of the encoded Petri net.

430 4.2. Constraint satisfaction problem

431 The following Boolean Constraint Satisfaction Problem (CSP) directly
432 derives from the above characterization:

433 **Definition 4.1.** Given a Petri net $\mathcal{P} = (P, T, W)$ encoding a Boolean net-
434 work $\mathcal{N} = (V, F)$. The CSP $\mathcal{C}(\mathcal{P})$ is the triple (R, D, C) where

- 435 • $R = P$, i.e., a variable is introduced for each place of \mathcal{P} ,
- 436 • $D(p) = \mathbb{B}$ for all $p \in R$, i.e., the variables are Boolean,
- 437 • $C = \{\neg p_v \vee \neg \bar{p}_v = 1 \mid \forall v \in V\} \wedge \{(p = 1 \rightarrow \bigvee_{p' \in \text{pred}(t)} p' = 1) \mid p \in$
438 $P, t \in \text{pred}(p)\}$.

439 **Proposition 4.1.** $\mathcal{C}(\mathcal{P})$ is satisfied by a valuation r if and only if

$$\{p \in P \mid r(p) = 1\}$$

440 is a conflict-free siphon of \mathcal{P} .

441 *Proof.* By the former part $\neg p_v \vee \neg \bar{p}_v = 1$ of C , the conflict-freeness is imposed
442 because for any satisfiable valuation r , $r(p_v) = r(\bar{p}_v) = 1$ is impossible for all
443 $v \in V$. As shown in [17], the latter part of C can characterize the set of all
444 generic siphons of \mathcal{P} . Hence, we can conclude the proof.

445 □

446 In [17], the set of all siphons of a given Petri net is characterized by a sim-
447 ilar Boolean CSP except the conflict-freeness constraint. From the encoded
448 CSP, the set of all *minimal* siphons of the Petri net can be enumerated in the
449 set inclusion order. For enumerating siphons in the set inclusion order, the
450 proposed method by [17] uses the technique that labels directly the Boolean
451 variables with increasing value selection (i.e., to test first the absence, then
452 the presence of a place in the candidate solution). The method has two
453 implementations, one uses an iterated SAT procedure and the other uses
454 Constraint Logic Programming (CLP) with backtracking.

455 One natural question is that how to use the CSP-based method for enu-
456 merating all the maximal conflict-free siphons of a Petri net encoding a
457 Boolean network? Of course, the set of all conflict-free siphons of the Petri
458 net can easily be characterized by the CSP model presented in [17] along with
459 the additional constraint $\neg p_v \vee \neg \bar{p}_v = 1$, for each $v \in V$, which represents

the conflict-freeness. However, the main concern is to enumerate all the *maximal* ones, which is not trivial to adapt from the CSP-based method. By Proposition 4.1, the set of all maximal conflict-free siphons of \mathcal{P} can be enumerated in the (maximality) set inclusion order, by restarting the search each time a conflict-free siphon S is found, with the following additional constraint for disallowing any subset of that conflict-free siphon: $\bigvee_{p \notin S} p = 1$. For enumerating conflict-free siphons in the set inclusion order, we can use the same technique as used in [17] but with the opposite setting, i.e., labeling directly the Boolean variables with decreasing value selection. The correctness of this technique comes from the fact that once S is found, it is the conflict-free siphon of maximum cardinality among all the remaining feasible conflict-free siphons. Similar to [17], the newly CSP-based method can also be implemented with SAT and CLP solvers.

For the SAT-based method, however a more direct method is to use a MaxSAT solver. We construct a MaxSAT problem with the following hard clauses:

$$(\neg p_v \vee \neg \bar{p}_v), \forall v \in V$$

and

$$(\neg p \vee \bigvee_{p' \in \text{pred}(t)} p'), \forall p \in P, \forall t \in \text{pred}(p).$$

We set a soft clause for each variable of the CSP and then use a “minimal correction subset” blocking strategy, which will ensure set-inclusion maximality of the solutions. This is what is implemented in **Trappist** using the RC2 MaxSAT solver [34] available through the python-sat package⁵.

For the CLP-based method, we use TODO: ...

4.3. Answer set programming-based method

Another possible method is to translate the characterization shown in Subsection 4.1 into the ASP \mathcal{L} as follows. We introduce atom **p-v** (resp. **n-v**) to denote place p_v (resp. \bar{p}_v), $\forall v \in V$. The set of all atoms in \mathcal{L} is given as $\mathcal{A} = \bigcup_{v \in V} \{\mathbf{p-v}, \mathbf{n-v}\}$. For each pair (p, t) where $p \in P, t \in T, t \in \text{pred}(p)$, we translate the rule (1) into the ASP rule

$$\mathbf{a_1}; \dots ; \mathbf{a_k} :- \mathbf{a}.$$

⁵<https://pysathq.github.io/docs/html/api/examples/rc2.html>

488 where $\mathbf{a} \in \mathcal{A}$ is the atom representing place p and $\{\mathbf{a}_1, \dots, \mathbf{a}_k\} \subseteq \mathcal{A}$ is the
 489 set of atoms representing places in $\text{pred}(t)$. The rule (2) is translated into
 490 the ASP rule

$$:- \mathbf{p-v}, \mathbf{n-v}.$$

491 for each $v \in V$. This ASP rule guarantees that two places representing
 492 the same node in \mathcal{N} never belong to the same siphon of \mathcal{P} , representing
 493 the conflict-freeness. Naturally, a Herbrand model (see, e.g., [35]) of \mathcal{L} is
 494 equivalent to a conflict-free siphon of \mathcal{P} . To guarantee that a Herbrand
 495 model is also a stable model (an answer set), we need to add to \mathcal{L} the two
 496 choice rules

$$\{\mathbf{p-v}\}. \{\mathbf{n-v}\}.$$

497 for each $v \in V$. Note that the number of atoms of \mathcal{L} is only $2n$, whereas
 498 the ASP encoding shown in [7] has as many atoms as the number of prime-
 499 implicants of the Boolean network and that number might be exponential in
 500 n . In [8], there is an ASP characterization of trap spaces that does not rely
 501 on minimal DNFs either and thus seems very similar to our ASP encoding.
 502 Remarkably it only requires the DNF for the *activation* part, using the in-
 503 formation that it will only be used for locally-monotonic Boolean networks.
 504 We would therefore expect that, when available, it will have comparable per-
 505 formance on the ASP part (the ASP program would be approximately twice
 506 smaller, though redundancy is not always bad in that field), but can also
 507 avoid combinatorial explosion of the Petri net encoding for some formula
 508 where the activation DNF is simple but the inhibition is not. Since `mpbn` is
 509 included in our benchmark this will be evaluated in our experiments.

510 Now, a solution (simply an answer set) $A \subseteq \mathcal{A}$ of \mathcal{L} is equivalent to a
 511 conflict-free siphon S of \mathcal{P} , thus a trap space m of \mathcal{N} . The conversion from A
 512 to m is straightforward. If $\mathbf{p-v} \in A$ then $v \in D_m$ and $m(v) = 0$. Conversely,
 513 if $\mathbf{n-v} \in A$ then $v \in D_m$ and $m(v) = 1$. Otherwise, $v \notin D_m$. Comput-
 514 ing multiple answer sets is built into ASP solvers and the solving collection
 515 POTASSCO [35] also features the option to find set-inclusion maximal an-
 516 swer sets with respect to the set of atoms. Naturally, a set-inclusion maximal
 517 answer set of \mathcal{L} is equivalent to a maximal conflict-free siphon of \mathcal{P} , thus a
 518 minimal trap space of \mathcal{N} . By using this built-in option, we can compute all
 519 the set-inclusion maximal answer sets of \mathcal{L} (resp. all the minimal trap spaces
 520 of \mathcal{N}) in one execution.

521 4.4. Integer linear programming-based method

522 We first show how an Integer Linear Programming (ILP) \mathcal{I} can define
 523 a set of all conflict-free siphons of the encoded Petri net \mathcal{P} . We introduce
 524 *binary* variable $\mathbf{p-v}$ (resp. $\mathbf{n-v}$) to denote place p_v (resp. \bar{p}_v), $\forall v \in V$. The
 525 set of all binary variables in \mathcal{I} is $\bigcup_{v \in V} \{\mathbf{p-v}, \mathbf{n-v}\}$. For each pair (p, t) where
 526 $p \in P, t \in T, t \in \text{pred}(p)$, we translate the rule (1) into the ILP inequality

$$\mathbf{a} \leq \mathbf{a_1} + \dots + \mathbf{a_k}$$

527 where \mathbf{a} is the binary variable representing place p and $\{\mathbf{a_1}, \dots, \mathbf{a_k}\}$ is the
 528 set of binary variable representing places in $\text{pred}(t)$. The rule (2) is translated
 529 into the ILP inequality

$$\mathbf{p-v} + \mathbf{n-v} \leq 1$$

530 for each $v \in V$. This inequality forbids both $\mathbf{p-v}$ and $\mathbf{n-p}$ receive the value
 531 1, thus representing the conflict-freeness. Since we only consider feasible
 532 solutions, the objective function is set to $\max \mathbf{p-v}$ for some $v \in V$. Naturally,
 533 a solution I of \mathcal{I} is equivalent to a conflict-free siphon S of \mathcal{P} . The conversion
 534 is that

$$S = \{p \in P \mid I(\mathbf{a-p}) = 1\}$$

535 where $\mathbf{a-p}$ is the binary variable presenting place p .

536 We can see the similarity between \mathcal{I} and the encoded ASP shown in the
 537 previous subsection. However, due to the nature of solutions of an ILP, it is
 538 hard to compute all the set-inclusion maximal solutions of \mathcal{I} in one execution
 539 of an ILP solver. Hence, we propose an iterative approach as follows.

540 The conflict-free siphon of maximum cardinality is of course maximal.
 541 Therefore, we impose the following objective function:

$$\max \sum_{v \in V} (\mathbf{p-v} + \mathbf{n-v}).$$

542 Now, \mathcal{I} can be solved using a general purpose ILP solver. If it admits any so-
 543 lution I^* , the corresponding conflict-free siphon (say S^*) is maximal. Hence,
 544 it makes sense that it does not need to find any other conflict-free siphon
 545 of the net that is strictly contained in S^* . To do this, we add to \mathcal{I} a new
 546 inequality

$$1 \leq \sum_{p \in P \setminus S^*} \mathbf{a-p}$$

547 where $\mathbf{a-p}$ is the binary variable presenting place p . Now, we solve \mathcal{I} again to
 548 find a new solution. If a new solution I' exists, then let S' be its corresponding
 549 conflict-free siphon. Indeed, abide by the newly added inequality, we have
 550 $S' \cap (P \setminus S^*) \neq \emptyset$ because there is some $\mathbf{a-p}$ with $p \in P \setminus S^*$ such that
 551 $I'(\mathbf{a-p}) = 1$. This implies that it is impossible that $S' = S^*$ or $S' \subset S^*$.
 552 By the objective function, it means that S' is the conflict-free siphon of
 553 maximum cardinality among the conflict-free siphons that are not contained
 554 in S^* . Hence, S' is also a maximal conflict-free siphon. Again, we add to \mathcal{I}
 555 a new inequality with respect to the newly found siphon. The above process
 556 is iterated until \mathcal{I} becomes unfeasible, this means that there is no further
 557 maximal conflict-free siphon. Thus, all the maximal conflict-free siphons of
 558 the Petri net have been found. We use ... as the ILP solver.

559 4.5. Computation of special types of trap spaces

560 In the field of systems biology, biologists may want to compute more
 561 special types of trap spaces beyond minimal trap spaces [19]. We shall show
 562 that our proposed methods can be easily adjusted to compute popular types
 563 of trap spaces. We illustrate the adjustments via the ASP-based method (see
 564 Subsection 4.3), but these adjustments are completely applicable for other
 565 approaches such as MaxSAT, CLP, and ILP.

566 First, the work by [36] uses the concept of stable motifs to build the suc-
 567 cession diagram of a Boolean network, a summary of the decisions in the
 568 network dynamics that lead to successively more restrictive nested stable
 569 motifs. The succession diagram is useful for control and decision making
 570 on this Boolean network. In particular, the proposed control methods are
 571 independent to the update scheme. It has been shown that a stable motif of
 572 a Boolean network is equivalent to a maximal trap space of this Boolean net-
 573 work [36]. Hence, it is necessary to develop an efficient method for computing
 574 maximal trap spaces of a Boolean network. We shall show how to adjust the
 575 ASP-method presented in Subsection 4.3 to compute maximal trap spaces.

576 We first provide the definition of maximal trap spaces. Let ε be the special
 577 trap space of \mathcal{N} where all the nodes are free. Of course, ε corresponds to the
 578 special conflict-free siphon \emptyset . A trap space m is called maximal if $m \neq \varepsilon$ and
 579 there is no other trap space m' such that $m' \neq \varepsilon$ and $m < m'$. Analogously,
 580 a conflict-free siphon S is called minimal if $S \neq \emptyset$ and there is no other
 581 trap space S' such that $S' \neq \emptyset$ and $S' \subset S$. By using the reasoning similar
 582 to the proof of Theorem 3.2, we can easily conclude that a maximal trap
 583 space of \mathcal{N} is equivalent to a minimal conflict-free siphon of its encoded

584 Petri net \mathcal{P} . Let \mathcal{L} be the ASP characterizing all conflict-free siphons of \mathcal{P}
 585 (see Subsection 4.3). Naturally, we need to exclude \emptyset from the solution space
 586 of \mathcal{L} (equivalently exclude ε from the set of trap spaces). To do this, we add
 587 to \mathcal{L} the ASP rule

$$\text{p-v}_1; \text{n-v}_1; \dots; \text{p-v}_n; \text{n-v}_n.$$

588 that ensures that every answer set of \mathcal{L} cannot be empty. Then a set-inclusion
 589 minimal answer set of \mathcal{L} is equivalent to a minimal conflict-free siphon of \mathcal{P} ,
 590 thus a maximal trap space of \mathcal{N} .

591 Second, we consider fixed points in Boolean networks. Let s be a fixed
 592 point of a Boolean network \mathcal{N} . We have a subspace m corresponding to s
 593 as follows: $\forall v \in V, m(v) = s(v)$, i.e., all nodes are fixed in m . Clearly, s is
 594 a trap set of \mathcal{N} regardless of the update scheme. Hence, m is a trap space
 595 of \mathcal{N} . In addition, since $|S_{\mathcal{N}}[m]| = 1$, m is also a minimal trap space. To
 596 compute all fixed points of \mathcal{N} , we can add more constraints to the encoded
 597 ASP characterizing all conflict-free siphons (equivalently trap spaces). For
 598 every $v \in V$, we add to the encoded ASP the rule

$$\text{p-v} ; \text{n-v}.$$

599 that ensures that for every conflict-free siphon S , it contains either p-v or n-v
 600 for every $v \in V$. Equivalently, the trap space corresponding to S is always
 601 a fixed point. Now, the set of answer sets of the encoded ASP is equivalent
 602 to the set of fixed points of \mathcal{N} . In particular, when solving the encoded ASP
 603 using an ASP solver, we do not need to use the built-in option for computing
 604 set-inclusion maximal answer sets. Note that we can also build another ASP
 605 characterizing all fixed points of \mathcal{N} based on the equivalence between a fixed
 606 point of \mathcal{N} and a deadlock of its Petri net encoding [21]. This approach may
 607 give a more compact ASP.

608 Third, we consider the trap spaces intersecting a given subspace m^* of
 609 a Boolean network. A trap space m intersects m^* if and only if $S_{\mathcal{N}}[m] \cap$
 610 $S_{\mathcal{N}}[m^*] \neq \emptyset$. It follows that for every v , if $m^*(v) = 0$ then $m(v) = 0$ or
 611 $m(v) = \star$, if $m^*(v) = 1$ then $m(v) = 1$ or $m(v) = \star$. For the former case, we
 612 add to \mathcal{L} the ASP rule

$$:- \text{n-v}.$$

613 that ensures that $m(v)$ cannot be 1. For the latter case, we add to \mathcal{L} the
 614 ASP rule

$$:- \text{p-v}.$$

615 that ensures that $m(v)$ cannot be 0. Now \mathcal{L} characterizes all trap spaces that
 616 intersect m^* .

617 Finally, we consider the trap spaces that are inside a given subspace m^*
 618 of a Boolean network. We first adjust \mathcal{L} to characterize all such trap spaces.
 619 A trap space m is inside m^* if and only if $m(v) = m^*(v)$ for every $v \in D_{m^*}$.
 620 If $m^*(v) = 0$, we add to \mathcal{L} the ASP rule

$$\mathbf{p-v.}$$

621 that ensures that $m(v) = 0$. If $m^*(v) = 1$, we add to \mathcal{L} the ASP rule

$$\mathbf{n-v.}$$

622 that ensures that $m(v) = 1$. It is noted that if we want to compute maximal
 623 trap spaces inside m^* , we need to exclude the conflict-free siphon correspond-
 624 ing m^* from the solution space. Specifically, we need to add to \mathcal{L} the ASP
 625 rule

$$\mathbf{p-v_{i1}; n-v_{i1}; \dots; p-v_{ik}; n-v_{ik}.}$$

626 where $\{v_{i_1}, \dots, v_{i_k}\}$ is the set of free nodes of m^* . This rule ensures that
 627 $m \neq m^*$. In the case that $m^* = \varepsilon$, we have all maximal trap spaces of the
 628 original Boolean network.

629 4.6. Inconsistent and incomplete data

630 So far we have assumed that we are always able to derive complete and
 631 consistent truth-tables (i.e., Boolean functions) that correctly capture the
 632 behavior of each node in a Boolean network. However, in practice it is rarely
 633 the case that a Boolean network of a biological system is fully understood
 634 and indeed, this is one important reason for modeling such networks. The
 635 data provided may be incomplete in the sense that information is missing
 636 about what happens in certain states, or it may be inconsistent in that we
 637 have conflicting information. The result is that the behavior of some nodes
 638 under certain conditions may be unknown [37].

639 Such incomplete and/or inconsistent behavioral information is problem-
 640 atic for Boolean networks with the synchronous update scheme, which are
 641 unable to represent the possibility of more than one next state. It can be
 642 addressed by using Boolean networks with asynchronous update schemes be-
 643 cause of their non-determinism. However, multiple update functions may be
 644 considered for each node, which is problematic for existing analysis methods

in Boolean networks. For example, if there are k nodes with two possible Boolean functions, then the analysis methods such as `PyBoolNet` and `mpbn` need to consider 2^k possible Boolean networks due to they inputs are Boolean networks with complete Boolean functions. Moreover, one interesting question maybe raised from systems biologists is if there is a common behavioral property (e.g., minimal trap space, attractor) among all possible combinations. To answer this question, the existing analysis methods may consider 2^k possible Boolean networks as well.

We shall show how our Petri net approach presented the previous section can handle the problem of inconsistent and incomplete data. The idea is to build the Petri net encoding of the Boolean network from its partial truth tables. More specifically, for each unknown-output row in the truth table of a node, we create two conflicting transitions representing the two possible outputs (i.e., 0 or 1). For every remaining row, we create transitions as usual following the Petri net encoding [21]. Because of the non-deterministic choice mechanisms of Petri nets, the new Petri net is still equivalent in dynamics to the incomplete Boolean network. One notable advantage of the connection between trap spaces of a Boolean network and conflict-free siphons of its Petri net encoding is that it still holds if the Petri net encoding changes (e.g., more or less transitions) but still maintain the equivalence in dynamics to the Boolean network. Hence, the minimal trap spaces of the incomplete Boolean network can be computed by computing maximal conflict-free siphons of the resulting Petri net. As a consequence, we can easily answer the question on common behavioral property.

Example 4.1. Consider an incomplete Boolean network $\mathcal{N} = (V, F)$ (mentioned in [37]), where $V = (g_1, g_2, g_3)$ and $F = (f_1, f_2, f_3)$ with $f_1 = g_2, f_3 = \neg g_1$. Herein, f_2 is an incomplete function and the partial truth table is as follows.

g_1	g_3	g_2
0	0	0
0	1	0
1	0	?
1	1	1

We have two possible update functions for g_2 : $f_2 = g_1 \wedge g_3$ for the case that $? = 0$ and $f_2 = g_1$ for the case that $? = 1$. Accordingly, we have

676 two possible Boolean networks. The first one has one minimal trap space:
 677 001. The second one has two minimal trap spaces: 001 and 110. Both
 678 have a common minimal trap space: 001. The asynchronous dynamics of
 679 the incomplete Boolean network is shown in Figure 2a. The result Petri net
 680 following our idea is given in Figure 2b. Herein, $t_{g_2}^3$ and $t_{g_2}^4$ are the two
 681 conflicting transitions representing the unknown-output row. This Petri net
 682 has only one maximal conflict-free siphon $\{p_{g_1}, p_{g_2}, \bar{p}_{g_3}\}$ corresponding to the
 683 common minimal trap space 001.

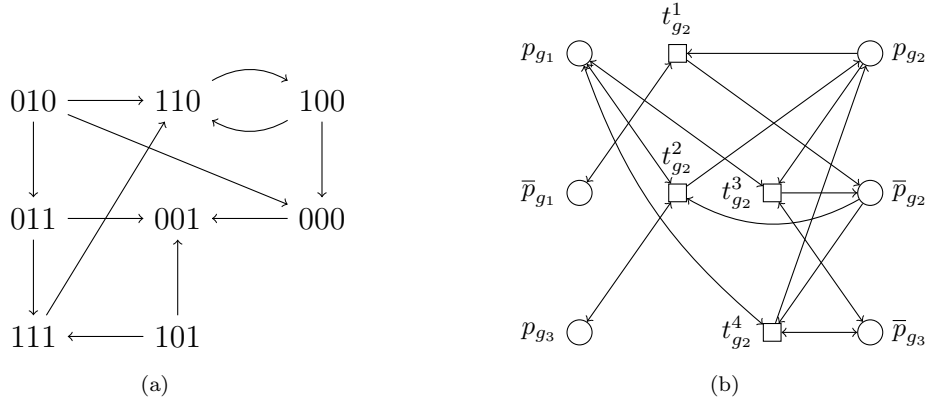


Figure 2: Asynchronous dynamics (self arcs are omitted for simplicity) and new Petri net encoding (the transitions corresponding to g_1 and g_3 are omitted for simplicity) of the incomplete Boolean network of Example 4.1

684 In summary, we can still perform meaningful analysis on the resulting
 685 Petri net. Note that the new Petri net may have more transitions, however
 686 the number of additional transitions is only $O(2k)$ where k is the number
 687 of unknown-output rows in the truth tables. This is actually a significant
 688 advantage, since we do not need to consider 2^k possible Boolean networks.
 689 It is worth noting that the approach proposed in [37] only deals with the
 690 case that $k = 1$. Furthermore, as more data becomes available for the un-
 691 derlying biological system, the Petri net model can be refined to reduce the
 692 amount of non-determinism (i.e., the number of conflicting transitions) it
 693 contains. Hence, Petri nets provide an interesting means of documenting the
 694 development of knowledge about a biological system.

695 5. Motivating example

696 For a few years now we have been collaborating with biologists who build
697 very large detailed and annotated maps and now wish to analyze the dy-
698 namics of the corresponding models. One of the main maps studied this way
699 represents knowledge about the Rheumatoid Arthritis [38], and was the main
700 motivation for the development of a tool to automatically transform it into
701 an executable Boolean network [6]. In the supplementary material of the pa-
702 per, an excerpt of the map, focused around the apoptosis (cell death) module
703 is transformed into a model of *reasonable* size, namely 180 Boolean variables
704 (model `F5_RA_apoptosis_executable_module.sbml` of supplementary ma-
705 terial S3, and model “RA-apoptosis” of Section 6). The study of such model,
706 though, is a big hurdle. Indeed, as stated in the article about another model
707 of the same size: “*The size of the CaSQ-inferred MAPK model (181 nodes)*
708 *made the calculation of stable states a non-realistic endeavour.*”

709 In practice, even if there is a huge number of attractors in such a model,
710 obtaining a sample of those can reveal very useful to invalidate the model and
711 lead to further refinement. In particular, it provides a feature-rich alternative
712 to random simulations for this type of very non-deterministic model. Being
713 able to detect that there are inconsistencies with published experimental data
714 in some of the first 1000 attractors, for instance, can lead to a much quicker
715 Systems Biology loop: model, invalidate, refine.

716 However, using a state-of-the-art tool like PyBoolNet [7] on that model ac-
717 tually fails at the phase of prime-implicant generation. `mpbn` [9] does not give
718 any answer either because it recognizes that model as non-locally-monotonic.
719 And hence, it is not possible to extract any (complex) attractor at all. This
720 is also true for the Alzheimer model also mentioned in that same article
721 and originally from [39] (F4 file in the original supplementary material, and
722 “Alzheimer” in Table 3), but actually not for the MAPK model for which the
723 first trap spaces can be obtained in reasonable time. The current practice
724 usually revolves then around fixing some source nodes to plausible values and
725 reducing the model accordingly. While this approach makes sense, it relies
726 on potentially arbitrary decisions, and *hides away* critical modelling choices
727 that were actually not part of the original Boolean network or even of the
728 starting map.

729 Using the method presented above, it is possible to convert the model to
730 PNML in about one second and to obtain the first 1000 minimal trap spaces
731 (including ones that contain more than one state) in a few milliseconds.

Unfortunately since this was not available at the time, the analysis of the model remained very high-level and qualitative, instead of being able to use the rich information of computed minimal trap spaces.

6. Evaluation

To evaluate the performance of the newly proposed methods (implemented as a Python package named `Trappist`) and the state-of-the-art methods (`bioLQM`⁶, `PyBoolNet` [7, 19], and `mpbn` [9]), we compared them on both `PyBoolNet`’s own model repository and many real-world models from various sources in the literature. It is worth noting that `mpbn` [9] only handles locally-monotonic models, whereas the other methods can handle general models. To obtain a more comprehensive comparison, we also used random models generated by a third-party software (i.e., `BoolNet R` package [29]). As explained in Section 5, in our benchmarks, we only searched for the first 1000 minimal trap spaces for each model.

To solve the ASP problems, we used the same ASP solver `Clingo` [35] and the same configuration as that used in `PyBoolNet` [7, 19] and `mpbn` [9]. Specifically, we used the configuration `-heuristic=Domain -enum-mod=domRec -dom-mod=3` (subset maximality, equivalent to the deprecated `--dom-pref=32 --heuristic=domain --dom-mod=7` used by `PyBoolNet`). We ran all the benchmarks on a machine whose environment is CPU: Intel® Core™ i9-11950H 2.60GHz \times 16, 16 GB DDR4 RAM, Ubuntu 20.04.5 LTS. Finally, we set a time limit of three minutes for each model.

All the models and a Jupyter notebook realizing the benchmarks can be found at <https://github.com/soli/trap-spaces-as-siphons>. These can be run on a Docker image in the cloud by clicking the “Binder” button.

6.1. *PyBoolNet* repository

Table 1 shows the experimental results on the models from the official `PyBoolNet` repository⁷. Column n denotes the number of nodes of each model. Column $|M|$ denotes the number of minimal trap spaces and for each method is given the computation time in seconds, asking only for the first 1000 trap spaces. A number in bold indicates a ratio greater than three compared to the best result. “NM” indicates a non-locally-monotonic

⁶<http://colomoto.org/biolqm/doc/tools-trap-space.html>

⁷<https://github.com/hklarner/pyboolnet/tree/master/pyboolnet/repository>

Table 1: Timing comparisons (in seconds) between **bioLQM** (LQM), **PyBoolNet** (PBN), **mpbn** and the four variants of **Trappist** on the **PyBoolNet** repository.

model	n	$ M $	LQM	PBN	mpbn	Trappist			
						SAT	CLP	ILP	ASP
1 arellano_rootstem	9	4	0.13	0.01	0.00	0.00	-	-	0.01
2 calzone_cellfate	28	27	0.12	0.02	0.01	0.01	-	-	0.01
3 dahlhaus_neuroplastoma	23	32	0.11	0.03	0.01	0.01	-	-	0.01
4 davidich_yeast	10	12	0.11	0.02	0.01	0.01	-	-	0.01
5 dinwoodie_life	15	7	0.11	0.01	0.00	0.01	-	-	0.01
6 dinwoodie_stomatal	13	1	0.10	0.01	0.00	0.00	-	-	0.01
7 faure_cellcycle	10	2	0.11	0.02	0.01	0.01	-	-	0.01
8 grieco_mapk	53	18	0.19	0.03	0.02	0.03	-	-	0.02
9 irons_yeast	18	1	0.12	0.03	0.01	0.01	-	-	0.02
10 jaoude_thdiff	103	> 1000	N/A	0.85	0.45	0.56	-	-	0.09
11 klamt_tcr	40	8	0.11	0.01	0.01	0.01	-	-	0.02
12 krumsiek_myeloid	11	6	0.10	0.01	0.00	0.00	-	-	0.01
13 multivalued	13	4	0.10	0.01	0.00	0.00	-	-	0.01
14 n12c5	11	5	0.11	17.83	0.01	0.01	-	-	0.01
15 n3s1c1a	2	2	0.10	0.01	0.00	0.00	-	-	0.01
16 n3s1c1b	2	2	0.09	0.02	0.00	0.00	-	-	0.01
17 n5s3	4	3	0.10	0.02	NM	0.00	-	-	0.01
18 n6s1c2	5	3	0.10	0.02	0.00	0.00	-	-	0.01
19 n7s3	6	3	0.11	0.02	0.00	0.00	-	-	0.01
20 raf	3	2	0.10	0.01	0.00	0.00	-	-	0.01
21 randomnet_n15k3	15	3	0.10	0.02	NM	0.01	-	-	0.01
22 randomnet_n7k3	7	10	0.10	0.01	NM	0.00	-	-	0.01
23 remy_tumorigenesis	34	25	0.15	0.94	0.02	0.02	-	-	0.02
24 saadatpour_guardcell	13	1	0.10	0.06	0.00	0.00	-	-	0.02
25 selvaggio_emt	56	> 1000	N/A	0.48	0.28	0.28	-	-	0.09
26 tournier_apoptosis	12	3	0.10	0.01	0.00	0.00	-	-	0.01
27 xiao_wnt5a	7	4	0.10	0.01	0.00	0.00	-	-	0.01
28 zhang_tlgl	60	156	0.60	0.09	0.09	0.07	-	-	0.04
29 zhang_tlgl_v2	60	258	0.64	0.04	0.08	0.11	-	-	0.04

model. There are four variants of **Trappist**: SAT (i.e., the MaxSAT-based method shown in Subsection 4.2), CLP (i.e., the CLP-based method shown in Subsection 4.2), ILP (i.e., the ILP-based method shown in Subsection 4.4), and ASP (i.e., the ASP-based method shown in Subsection 4.3).

As shown in Table 1, for most of the models of the **PyBoolNet** repository, the results are comparable with all minimal trap spaces found very fast.

For 5 of the 29 models, `mpbn` did not give any answer because it recognized these models as not locally-monotonic. Note that on some very small models, `Trappist` is sometimes slower than `PyBoolNet` and/or `mpbn`, but still significantly under one second. On the contrary, on every model that was a bit challenging for `PyBoolNet` or `mpbn`, the new method is far more efficient with speedups between one and two orders of magnitude.

6.2. BBM repository

Currently, a research group has made a great effort for building a collection (called BBM) of real-world Boolean models from various sources used in systems biology. It aims to be a comprehensive collection suitable for benchmarking and testing new tools and methods. It is released and maintained at <https://github.com/sybila/biodivine-boolean-models>. We here tested all the compared methods on this model repository.

Table 2: Results on the real-world models from the BBM repository.

Method	# failures	avg-lqm (s)	avg-mono (s)	avg-all (s)
bioLQM	9 (134)	12.87	N/A	N/A
PyBoolNet	12	8.87	11.00	13.59
mpbn	2 (187)	N/A	2.31	N/A
Trappist-MaxSAT	1	0.03	1.09	1.01
Trappist-CLP	-	-	-	-
Trappist-ILP	-	-	-	-
Trappist-ASP	1	0.05	1.02	0.93

Table 2 shows the experimental results on the 211 real-world models from the BBM repository. Column 2 expresses the numbers of failures (i.e., did not finish the computation within a time limit of three minutes) of each method. For the case of `bioLQM`, we only considered the models that have at most 1000 minimal trap spaces. The number of such models is 134 (per all 211 models) and is denoted inside the parentheses. For the case of `mpbn`, we only considered the models that are locally-monotonic. The number of such models is 187 (per all 211 models) and is denoted inside the parentheses. Columns 3-5 express the average running time (in seconds) of each method for the models having at most 1000 minimal trap spaces, the locally-monotonic models, and all the models, respectively. Note that when computing the

794 average running time, if the running time exceeds 180s, it is considered as
795 180s. From the results shown in Table 2, we reported several observations as
796 follows.

797 *6.3. Selected models*

798 We used a set of real-world Boolean networks lying in various scales col-
799 lected from numerous bibliographic sources. These models are quite big
800 (in size), complex (i.e., having high average in-degree, which is related to
801 the number of prime-implicants) and most of them have never been fully
802 analyzed. Note that these models are not included in the `PyBoolNet` and
803 `BBM` repositories. We then applied `bioLQM`, `PyBoolNet`, `mpbn`, and the four
804 variants of `Trappist` to computing minimal trap spaces of these real-world
805 models. It is notable that unlike existing analysis shown in the literature, we
806 did not fix specific values for source nodes in these models. Table 3 shows the
807 experimental results on those models. “DNF” means that the method did
808 not finish the computation (stopping at the first 1000 minimal trap spaces)
809 within the timeout of two minutes. In the case of `bioLQM`, “N/A” means that
810 the number of all minimal trap spaces of the model is larger than 1000 and
811 we did not recorded the running time of `bioLQM` because it always requires to
812 compute all minimal trap spaces. A number in bold indicates a ratio greater
813 than or equal to 10 compared to the best result. Other notations are similar
814 to those in Table 1. Hereafter, we analyze in detail the results with respect
815 to minimal trap space computation.

816 The first observation is that for 26 of the 33 models (more than 78%),
817 `mpbn` did not give any answer because it recognized that these models as
818 not locally-monotonic. For 6 of the 33 models where `mpbn` returned the
819 answers, `mpbn` and `Trappist` are comparable in computation time, though
820 surprisingly `mpbn` appears a bit slower on average. Note however that `mpbn`
821 was the only tool to provide a solution for the SN-5 model, thus confirming
822 that if the activation function is in the right form, not having to compute the
823 inactivation function’s disjunctive normal form can render a difficult problem
824 tractable. However, since `mbpn` can handle only locally-monotonic models
825 and `Trappist` can handle general models, it is difficult to further compare
826 between them. Hence, we focus on only comparisons between `PyBoolNet`
827 and `Trappist` in the following observations.

828 The second observation is that the proposed method vastly outperforms
829 `PyBoolNet` in computational time, on each and every model, and sometimes
830 with orders of magnitude of difference (e.g., for most models in the 100–1000

Table 3: Timing comparisons (in seconds) between **bioLQM** (LQM), **PyBoolNet** (PBN), **mpbn** and the four variants of **Trappist** on selected models from the literature.

model	n	$ M $	LQM	PBN	mpbn	Trappist			
						SAT	CLP	ILP	ASP
1 Arabidopsis_thaliana [40]	15	8	0.10	0.06	NM	0.01	-	-	0.02
2 Bcell [41]	73	72	0.23	0.04	0.08	0.06	-	-	0.05
3 Cacace_TdevModel [42]	61	28	1.29	5.67	NM	0.06	-	-	0.08
4 Corral_ThIL17diff [43]	92	> 1000	N/A	107.57	0.76	0.56	-	-	0.16
5 DNA_damage [40]	26	16	0.24	0.33	NM	0.02	-	-	0.05
6 Drosophila [44]	52	128	0.33	0.05	0.07	0.06	-	-	0.05
7 EMT [36]	69	268	39.22	1.01	0.20	0.12	-	-	0.05
8 mast_cell [6]	73	> 1000	N/A	0.09	0.55	0.37	-	-	0.15
9 FT-GRN [45]	23	32	DNF	DNF	NM	0.03	-	-	0.19
10 hedgehog [40]	65	> 1000	N/A	DNF	0.50	0.34	-	-	0.33
11 metastatic [40]	10	4	0.10	0.04	NM	0.01	-	-	0.02
12 Pancreatic_Cancer [40]	43	> 1000	N/A	0.11	0.36	0.17	-	-	0.06
13 Pluripotent [40]	36	276	0.37	0.43	NM	0.07	-	-	0.06
14 Rho-GTPases [40]	33	2	0.17	0.57	40.39	0.07	-	-	0.11
15 Pluripotency [46]	36	440	DNF	DNF	NM	0.16	-	-	0.28
16 p53_high_dna [40]	16	1	0.38	1.76	NM	0.08	-	-	0.14
17 p53_low_dna [40]	16	1	0.41	1.76	NM	0.07	-	-	0.14
18 angiofull [47]	142	> 1000	N/A	0.17	1.06	0.88	-	-	0.23
19 EMT_Mech [48]	136	82	DNF	14.01	0.27	0.20	-	-	0.25
20 EMT_Mech_TGFbeta [48]	150	492	DNF	11.28	0.78	0.69	-	-	0.35
21 MAPK [6]	181	> 1000	N/A	13.58	1.76	1.51	-	-	0.27
22 macrophage [40]	136	> 1000	N/A	0.54	1.09	0.84	-	-	0.27
23 RA_apoptosis [6]	180	> 1000	N/A	DNF	1.43	1.55	-	-	0.19
24 Adhesion_CIP [49]	121	78	56.81	4.25	0.23	0.17	-	-	0.19
25 angiogenesis [40]	141	> 1000	N/A	0.16	1.07	1.06	-	-	0.16
26 T-cell-co-receptor [40]	206	> 1000	N/A	DNF	1.52	2.26	-	-	0.35
27 Snf1-pathway [50]	202	> 1000	N/A	1.13	1.47	1.43	-	-	0.31
28 Mycobacterium [40]	317	> 1000	N/A	0.42	2.36	4.91	-	-	0.44
29 Leishmania [40]	342	> 1000	N/A	DNF	2.56	5.62	-	-	0.46
30 TcellCheckPoint [51]	218	> 1000	N/A	4.99	NM	1.96	-	-	0.28
31 Cholocystokinin [6]	383	> 1000	N/A	0.36	2.99	4.81	-	-	0.37
32 Alzheimer [6]	762	> 1000	N/A	DNF	NM	18.21	-	-	0.79

nodes size range). Note that for all the cases where **PyBoolNet** did not manage to finish before the timeout, as marked by “DNF” in Table 3, the timeout occurred during the computation of the prime-implicants. Hence, not even a single minimal trap space was output by that method. The computational advantage is therefore immediately a practical advantage since on the one

836 hand the state-of-the-art method did not allow any analysis whatsoever of
837 the models, and on the other hand the proposed method could provide, very
838 often under one second, the first thousand minimal trap spaces. For mod-
839 ellers having a critical look at a model and in a *model, invalidate, refine* loop
840 this means a huge difference in the models that are amenable to study.

841 Note that even with a very restricted time-limit of two minutes, it was
842 possible with the proposed technique to find *all* minimal trap spaces of small
843 models (roughly under 130 nodes, i.e., considered as quite big up to now).
844 Though it might seem impractical to handle tens of thousands of such pos-
845 sible complex attractors in a manual way, i.e., to compare them to specific
846 experimental conditions and corresponding data, we hope that an automatic
847 analysis of such attractors might become possible with systematic verification
848 methods, not unlike that described in [51]. Since the ASP code is declarative
849 by nature, it is also possible to add to it supplementary constraints coming
850 from the modeler in case one is looking for specific attractors. Finally, sam-
851 pling from the ASP-generated solutions as is done in [52] would allow for a
852 different type of exploration.

853 The third observation is that for all the models where `PyBoolNet` finished
854 before the timeout, once `PyBoolNet` went through the prime-implicant phase,
855 its ASP solving phase quickly returned the first 1000 minimal trap spaces, all
856 under one second. For these models, the ASP solving phase of the proposed
857 method also took very short time, all under one second. Hence, with the
858 experimental results shown in this paper, the practical differences between
859 our ASP encoding and that of `PyBoolNet` are not distinctly exposed. The
860 fact that our new ASP encoding is guaranteed to be linear in the number
861 of nodes of the original model does not seem to be crucial here, however a
862 much deeper analysis of those cases remains to be done.

863 The last observation is that for very large models (i.e., more than two
864 thousand nodes) the proposed method did not manage to finish the Petri net
865 conversion before the timeout, as marked by “???” in Table 3. This points to
866 the fact that our current choice of using a BDD-based translation to obtain
867 that Petri net encoding, though it provides a small/efficient ASP might be
868 too costly to handle the largest models. In such a case, a more *naive* encoding
869 might provide a much larger ASP program, with many redundant rules, but
870 easier/faster to obtain. The evaluation of the feasibility of such strategy,
871 and of its impact on smaller instances, remains to be done and is out of the
872 scope of this first article. Recognizing that a model is locally-monotonic and
873 applying in that specific case dedicated strategies as those of `mpbn` might also

874 be a partial solution as shown in the SN-5 case, but not for the turei-2016
875 model.

876 Note that though enumerating the extremal siphons of a Petri net is ex-
877 ponential (see [17] for instance) this is apparently not the bottleneck of the
878 proposed method, showing once again that networks obtained from biochem-
879 ical models do have a specific structure.

880 6.4. Randomly generated models

881 We randomly generated a set of N-K models [1] with network size n in the
882 set $\{100, 150, 200, 250, 300, 350, 400\}$ and $K = 3$ (i.e., each node has exactly
883 three input nodes). We chose N-K models because they are a useful tool for
884 studying the dynamics of Boolean networks [1, 7]). For each network size, 50
885 instances were generated using the `generateRandomNKNetwork` function. In
886 total, we have 350 random models. We then applied the compared methods
887 to these models and recorded the numbers of failures (i.e., failed to obtain
888 the result within a time limit of three minutes) as well as the average running
889 time (inside the parentheses) in each method for each network size n . It
890 worth noting that N-K models usually have small numbers of minimal trap
891 spaces [7]. Hence, we searched for all solutions in each model, which makes
892 the comparison to `bioLQM` more comprehensive. In addition, each node has
893 only three input nodes, i.e., the number of prime-implicants of the associated
894 Boolean function is small. Hence, `PyBoolNet` always passed the phase of
895 computing prime-implicants in every model even within 1s, which enables us
896 to compare the ASP encoding of `PyBoolNet` and that of `Trappist`.

Table 4: Results on N-K models.

n	LQM	mpbn	PBN	Trappist			
				SAT	CLP	ILP	ASP
100	50 (> 180)	50 (N/A)	0 (0.07)	0 (0.05)	- ()	- ()	0 (0.09)
150	50 (> 180)	50 (N/A)	0 (0.14)	0 (0.10)	- ()	- ()	0 (0.14)
200	50 (> 180)	50 (N/A)	0 (0.43)	0 (0.25)	- ()	- ()	0 (0.24)
250	50 (> 180)	50 (N/A)	0 (1.92)	0 (1.04)	- ()	- ()	0 (0.56)
300	50 (> 180)	50 (N/A)	0 (9.68)	0 (4.46)	- ()	- ()	0 (1.83)
350	50 (> 180)	50 (N/A)	1 (46.54)	0 (20.09)	- ()	- ()	0 (6.10)
400	50 (> 180)	50 (N/A)	29 (144.09)	12 (90.36)	- ()	- ()	1 (33.01)

897 Table 4 shows the experimental results on N-K models. Column n de-
898 notes the network size. Columns “LQM” and “PBN” show the results of
899 **bioLQM** and **PyBoolNet**, respectively. For each method, the number outside
900 the parentheses indicates the number of failures, whereas the number inside
901 the parentheses indicates the average running time (in seconds). Note that
902 when computing the average running time, if the running time exceeds 180s,
903 it is considered as 180s. From these results, we obtained several observations
904 consistent with those obtained for real-world models.

905 First, ...
906 Second, ...
907 Third, ...
908 Finally, ...

909 7. Conclusion

910 In this article we explored and proved for the first time the equivalence
911 between (minimal) trap spaces of a general Boolean network and (maximal)
912 conflict-free siphons of its Petri net encoding. From this equivalence, we pro-
913 posed a new approach for the computation of minimal trap spaces of Boolean
914 networks, based on the enumeration of maximal conflict-free siphons of Petri
915 nets. We also proposed the four possible methods using MaxSAT, CLP, ILP,
916 and ASP for implementing the new approach. The proposed methods were
917 evaluated on many real-world models from the literature and randomly gen-
918 erated models. The experimental results show that ... We believe that this
919 opens up the way to a much better analysis of large Boolean networks, which
920 is needed with the advent of automatic model-generation pipelines [53].

921 Second, the experimental results shown in this paper mostly expose the
922 differences caused by the prime-implicant phase of **PyBoolNet**. There is
923 much more information required to distinctly study the practical differ-
924 ences between our ASP encoding and that of **PyBoolNet**, and notably their
925 size/efficiency ratio. Hence, we plan to conduct experiments on more real-
926 world models or maybe random models that can be randomly generated by
927 using BoolNet [29]. Such experiments should include a time-course of the
928 number of ASP solutions found for proper comparison of the ASP encodings.

929 In addition, there are possibly other methods for computing maximal
930 conflict-free siphons in Petri nets, like SAT/MaxSAT approaches [17]. Al-
931 though these approaches do not directly support the maximal conflict-free
932 siphon computation now, we plan to investigate them in the future. They

933 could replace our ASP program if they outperform it. However, the current
 934 method appears to already perform very well even on the biggest models we
 935 have considered.

936 Finally, we think that the links between Petri nets and Boolean networks
 937 that we stumbled upon in this method might have deeper roots. Exploring
 938 those connections might lead both to interesting topics of research for Petri
 939 nets, like a notion of trap-spaces, and for Boolean networks.

940 References

- 941 [1] L. Glass, S. A. Kauffman, The logical analysis of continuous, non-linear
 942 biochemical control networks, *J. Theor. Biol.* 39 (1973) 103–129.
- 943 [2] R. Thomas, Boolean formalisation of genetic control circuits, *J. Theor.*
 944 *Biol.* 42 (1973) 565–583.
- 945 [3] R. Thomas, R. d’Ari, Biological feedback, CRC press, 1990.
- 946 [4] R. Thomas, Regulatory networks seen as asynchronous automata: a
 947 logical description, *J. Theor. Biol.* 153 (1991) 1–23.
- 948 [5] R.-S. Wang, A. Saadatpour, R. Albert, Boolean modeling in systems
 949 biology: an overview of methodology and applications, *Phys. Biol.* 9
 950 (2012) 055001.
- 951 [6] S. S. Aghamiri, V. Singh, A. Naldi, T. Helikar, S. Soliman, A. Niarakis,
 952 J. Xu, Automated inference of Boolean models from molecular interac-
 953 tion maps using CaSQ, *Bioinform.* 36 (2020) 4473–4482.
- 954 [7] H. Klarner, A. Bockmayr, H. Siebert, Computing maximal and minimal
 955 trap spaces of Boolean networks, *Nat. Comput.* 14 (2015) 535–544.
- 956 [8] S. Chevalier, C. Froidevaux, L. Paulevé, A. Y. Zinovyev, Synthesis of
 957 Boolean networks from biological dynamical constraints using answer-
 958 set programming, in: *International Conference on Tools with Artificial*
 959 *Intelligence*, IEEE, 2019, pp. 34–41.
- 960 [9] L. Paulevé, J. Kolčák, T. Chatain, S. Haar, Reconciling qualitative,
 961 abstract, and scalable modeling of biological networks, *Nat. Commun.*
 962 11 (2020) 1–7.

- 963 [10] M. Noual, D. Regnault, S. Sené, About non-monotony in Boolean au-
964 tomata networks, *Theor. Comput. Sci.* 504 (2013) 12–25.
- 965 [11] J. L. Peterson, *Petri net theory and the modeling of systems*, Prentice
966 Hall PTR, 1981.
- 967 [12] T. Murata, Petri nets: Properties, analysis and applications, *Proc.*
968 *IEEE* 77 (1989) 541–580.
- 969 [13] V. N. Reddy, M. L. Mavrovouniotis, M. N. Liebman, Petri net rep-
970 resentations in metabolic pathways, in: *International Conference on*
971 *Intelligent Systems for Molecular Biology*, AAAI, 1993, pp. 328–336.
- 972 [14] I. Zevedei-Oancea, S. Schuster, Topological analysis of metabolic net-
973 works based on Petri net theory, *Silico Biol.* 3 (2003) 323–345.
- 974 [15] M. A. Blätke, M. Heiner, W. Marwan, Biomodel engineering with Petri
975 nets, in: *Algebraic and Discrete Mathematical Methods for Modern*
976 *Biology*, Elsevier, 2015, pp. 141–192.
- 977 [16] O. Oanea, H. Wimmel, K. Wolf, New algorithms for deciding the siphon-
978 trap property, in: *International Conference on Applications and Theory*
979 *of Petri Nets*, Springer, 2010, pp. 267–286.
- 980 [17] F. Nabli, T. Martinez, F. Fages, S. Soliman, On enumerating mini-
981 mal siphons in Petri nets using CLP and SAT solvers: theoretical and
982 practical complexity, *Constraints An Int. J.* 21 (2016) 251–276.
- 983 [18] V. Trinh, B. Benhamou, K. Hiraishi, S. Soliman, Minimal trap spaces of
984 logical models are maximal siphons of their Petri net encoding, in: *In-*
985 *ternational Conference on Computational Methods in Systems Biology*,
986 Springer, 2022, pp. 158–176.
- 987 [19] H. Klarner, A. Streck, H. Siebert, PyBoolNet: a python package for the
988 generation, analysis and visualization of Boolean networks, *Bioinform.*
989 33 (2017) 770–772.
- 990 [20] L. C. Fontanals, E. Tonello, H. Siebert, Control strategy identification
991 via trap spaces in Boolean networks, in: *International Conference on*
992 *Computational Methods in Systems Biology*, Springer, 2020, pp. 159–
993 175.

994 [21] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, S. Schwoon, Characteriza-
 995 tion of reachable attractors using Petri net unfoldings, in: International
 996 Conference on Computational Methods in Systems Biology, Springer,
 997 2014, pp. 129–142.

998 [22] C. Chaouiya, E. Remy, P. Ruet, D. Thieffry, Qualitative modelling of
 999 genetic networks: From logical regulatory graphs to standard Petri nets,
 1000 in: International Conference on Applications and Theory of Petri Nets,
 1001 Springer, 2004, pp. 137–156.

1002 [23] C. Chaouiya, A. Naldi, E. Remy, D. Thieffry, Petri net representation of
 1003 multi-valued logical regulatory graphs, *Nat. Comput.* 10 (2011) 727–750.

1004 [24] T. Chatain, S. Haar, J. Kolcák, L. Paulevé, A. Thakkar, Concurrency
 1005 in Boolean networks, *Nat. Comput.* 19 (2020) 91–109.

1006 [25] C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, et al., SBML
 1007 qualitative models: a model representation format and infrastructure to
 1008 foster interactions between qualitative modelling formalisms and tools,
 1009 *BMC Syst. Biol.* 7 (2013) 1–15.

1010 [26] S. M. Keating, D. Waltemath, M. König, F. Zhang, et al., SBML Level
 1011 3: an extensible format for the exchange and reuse of biological models,
 1012 *Mol. Syst. Biol.* 16 (2020) e9110.

1013 [27] C. Chaouiya, A. Naldi, D. Thieffry, Logical modelling of gene regulatory
 1014 networks with GINsim, in: *Bacterial Molecular Networks*, Springer,
 1015 2012, pp. 463–479.

1016 [28] A. Naldi, P. T. Monteiro, C. Müssel, C. for Logical Models, Tools,
 1017 H. A. Kestler, D. Thieffry, I. Xenarios, J. Saez-Rodriguez, T. Helikar,
 1018 C. Chaouiya, Cooperative development of logical modelling standards
 1019 and tools with CoLoMoTo, *Bioinform.* 31 (2015) 1154–1159.

1020 [29] C. Müssel, M. Hopfensitz, H. A. Kestler, BoolNet - an R package for
 1021 generation, reconstruction and analysis of Boolean networks, *Bioinform.*
 1022 26 (2010) 1378–1380.

1023 [30] D. Angeli, P. D. Leenheer, E. Sontag, A Petri net approach to persistence
 1024 analysis in chemical reaction networks, in: *Biology and Control Theory:
 1025 Current Challenges*, Springer, 2007, pp. 181–216.

- 1026 [31] D. Angeli, P. D. Leenheer, E. D. Sontag, Persistence results for chemical
1027 reaction networks with time-dependent kinetics and no global conserva-
1028 tion laws, *SIAM J. Appl. Math.* 71 (2011) 128–146.
- 1029 [32] E. Degrand, F. Fages, S. Soliman, Graphical conditions for rate inde-
1030 pendence in chemical reaction networks, in: *International Conference on*
1031 *Computational Methods in Systems Biology*, Springer, 2020, pp. 61–78.
- 1032 [33] G. Liu, K. Barkaoui, A survey of siphons in Petri nets, *Inf. Sci.* 363
1033 (2016) 198–220.
- 1034 [34] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient MaxSAT
1035 solver, *J. Satisf. Boolean Model. Comput.* 11 (2019) 53–64.
- 1036 [35] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub,
1037 M. Schneider, Potassco: The Potsdam answer set solving collection,
1038 *AI Commun.* 24 (2011) 107–124.
- 1039 [36] J. C. Rozum, J. G. T. Zañudo, X. Gan, D. Deritei, R. Albert, Parity
1040 and time reversal elucidate both decision-making in empirical models
1041 and attractor scaling in critical Boolean networks, *Sci. Adv.* 7 (2021)
1042 eabf8124.
- 1043 [37] L. J. Steggles, R. Banks, O. Shaw, A. Wipat, Qualitatively modelling
1044 and analysing genetic regulatory networks: a Petri net approach, *Bioin-*
1045 *form.* 23 (2006) 336–343.
- 1046 [38] V. Singh, M. Ostaszewski, G. D. Kalliolias, G. Chiocchia, R. Olaso,
1047 E. Petit-Teixeira, T. Helikar, A. Niarakis, Computational systems bi-
1048 ology approach for the study of rheumatoid arthritis: from a molecular
1049 map to a dynamical model, *Genom. Comput. Biol.* 4 (2018) 100050.
- 1050 [39] S. Ogishima, S. Mizuno, M. Kikuchi, A. Miyashita, R. Kuwano,
1051 H. Tanaka, J. Nakaya, AlzPathway, an updated map of curated sig-
1052 naling pathways: towards deciphering Alzheimer’s disease pathogenesis,
1053 in: *Systems Biology of Alzheimer’s Disease*, Springer, 2016, pp. 423–432.
- 1054 [40] C. Kadelka, T.-M. Butrie, E. Hilton, J. Kinseth, H. Serdarevic, A meta-
1055 analysis of Boolean network models reveals design principles of gene
1056 regulatory networks, *arXiv preprint arXiv:2009.01216* (2020).

- 1057 [41] P. Dutta, L. Ma, Y. Ali, P. M. Sloom, J. Zheng, Boolean network model-
1058 ing of B-cell apoptosis and insulin resistance in type 2 diabetes mellitus,
1059 BMC Syst. Biol. 13 (2019) 1–12.
- 1060 [42] E. Cacace, S. Collombet, D. Thieffry, Logical modeling of cell fate
1061 specification—Application to T cell commitment, in: Current Topics in
1062 Developmental Biology, Elsevier, 2020, pp. 205–238.
- 1063 [43] K. F. Corral-Jara, C. Chauvin, W. Abou-Jaoudé, M. Grandclaudeon,
1064 A. Naldi, V. Soumelis, D. Thieffry, Interplay between SMAD2 and
1065 STAT5A is a critical determinant of IL-17A/IL-17F differential expres-
1066 sion, Mol. Biomed. 2 (2021) 1–16.
- 1067 [44] M. R. Vega, Analyzing toys models of Arabidopsis and Drosophila us-
1068 ing Z3 SMT-LIB, in: Independent Component Analyses, Compressive
1069 Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII,
1070 volume 9118, SPIE, 2014, pp. 240–254.
- 1071 [45] E. C. Chávez-Hernández, S. Quiroz, B. García-Ponce, E. R. Álvarez-
1072 Buyla, The flowering transition pathways converge into a complex gene
1073 regulatory network that underlies the phase changes of the shoot apical
1074 meristem in Arabidopsis thaliana, Front. Plant Sci. 13 (2022) 852047.
- 1075 [46] A. Yachie-Kinoshita, K. Onishi, J. Ostblom, M. A. Langley, E. Posfai,
1076 J. Rossant, P. W. Zandstra, Modeling signaling-dependent pluripotency
1077 with Boolean logic to predict cell fate transitions, Mol. Syst. Biol. 14
1078 (2018) e7952.
- 1079 [47] N. Weinstein, L. Mendoza, I. Gitler, J. Klapp, A network model to
1080 explore the effect of the micro-environment on endothelial cell behavior
1081 during angiogenesis, Front. Physiol. 8 (2017) 960.
- 1082 [48] E. Sullivan, M. Harris, A. Bhatnagar, E. Guberman, I. Zonfa, E. R. Re-
1083 gan, Boolean modeling of mechanosensitive Epithelial to Mesenchymal
1084 Transition and its reversal, bioRxiv (2022).
- 1085 [49] E. Guberman, H. Sherief, E. R. Regan, Boolean model of anchorage
1086 dependence and contact inhibition points to coordinated inhibition but
1087 semi-independent induction of proliferation and migration, Comput.
1088 Struct. Biotechnol. J. 18 (2020) 2145–2165.

- 1089 [50] T. Lubitz, N. Welkenhuysen, S. Shashkova, L. Bendrioua, S. Hohmann,
1090 E. Klipp, M. Krantz, Network reconstruction and validation of the
1091 Snf1/AMPK pathway in baker’s yeast based on a comprehensive litera-
1092 ture review, *npj Syst. Biol. Appl.* 1 (2015) 1–10.
- 1093 [51] C. Hernandez, M. Thomas-Chollier, A. Naldi, D. Thieffry, Computa-
1094 tional verification of large logical models—Application to the prediction
1095 of T cell response to checkpoint inhibitors, *Front. Physiol.* 11 (2020)
1096 558606.
- 1097 [52] S. Chevalier, V. Noël, L. Calzone, A. Y. Zinovyev, L. Paulevé, Synthesis
1098 and simulation of ensembles of Boolean networks for cell fate decision,
1099 in: *International Conference on Computational Methods in Systems Bi-*
1100 *ology*, Springer, 2020, pp. 193–209.
- 1101 [53] M. Ostaszewski, A. Niarakis, A. Mazein, I. Kuperstein, R. Phair,
1102 A. Orta-Resendiz, V. Singh, S. S. Aghamiri, M. L. Acencio, E. Glaab,
1103 et al., COVID19 Disease Map, a computational knowledge repository of
1104 virus–host interaction mechanisms, *Mol. Syst. Biol.* 17 (2021) e10387.