

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 its number of minimal trap spaces. This issue lim-
 41 its the efficiency of the BDD-based method. The study [10] highlights the
 42 need for non-locally-monotonic Boolean networks in both biological and the-
 43 oretical aspects. Hence, it is still necessary to develop efficient methods for
 44 computing minimal trap spaces of large-scale general Boolean networks.

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

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

on 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 Programming (CP), and Integer Linear Programming (ILP). Fourth, we discuss in detail how to compute several special types of trap spaces in a Boolean network. Besides minimal trap spaces, these special types also play crucial roles in analyzing and controlling Boolean networks [19]. Fifth, 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 extensive 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 dozens of representative real-world models), with more comprehensive insights are obtained.

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.

2.1. Boolean networks

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

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

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

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

$$v_i(t+1) = \begin{cases} f_i(v(t)) \\ \text{or} & v_i(t) \end{cases}$$

114 where $v(t)$ is the state of \mathcal{N} at time t . Note that for simplicity, we write
115 $f_i(v(t))$ even if $IN(v_i) \subsetneq V$ (i.e., $IN(v_i)$ does not contain some nodes of
116 V). An update scheme of a Boolean network specifies which nodes update
117 their states, as defined above, through time evolution [4]. Following the
118 update scheme, the Boolean network transits from a state to another state
119 (possibly identical). This transition is called the *state transition* and denoted
120 by $\rightarrow \subseteq \mathcal{S}_{\mathcal{N}} \times \mathcal{S}_{\mathcal{N}}$. Then the dynamics of \mathcal{N} is captured by the directed graph
121 $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ called the State Transition Graph (STG). There are many different
122 update schemes, but the two main types [4] are: synchronous, where all
123 the nodes are update simultaneously, and fully asynchronous, where only
124 one node is selected non-deterministically to be updated. I removed the
125 “random” part since actually, how the only node is selected is not strictly
126 part of the asynchronous update... Updating always the nodes in the order
127 in which they appear in the model for instance, though pretty bad, would
128 be an admissible async update run. Usually for such things where you could
129 even imagine an “opponent” choosing which node to update, you can say it’s
130 “non-deterministic” but not necessarily “random”.

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* [20, 21].

136 A 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 $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ [22]. An attractor of size 1 is called a fixed point,
 141 otherwise it is called a cyclic or complex attractor [7].

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

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

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

146 A *trap space* is defined as a subspace that is also a trap set. It is noted
 147 that trap spaces of a Boolean network are independent of the update scheme
 148 of this model [7]. Then, we define a partial order $<$ on $\mathcal{S}_{\mathcal{N}}^{\star}$ as: $m < m'$ if and
 149 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
 150 is minimal if and only if there is no trap space $m' \in \mathcal{S}_{\mathcal{N}}^{\star}$ such that $m' < m$.

151 For example, let us consider the Boolean network shown in Example 2.1.
 152 Figure 1(a) shows the dynamics of this model under the fully asynchronous
 153 update (i.e., only one node is updated at each time step). The model has all
 154 trap spaces, $m_1 = 11$ and $m_2 = \star\star$. Since $m_1 < m_2$, m_1 is the only minimal
 155 trap space of the Boolean network.

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

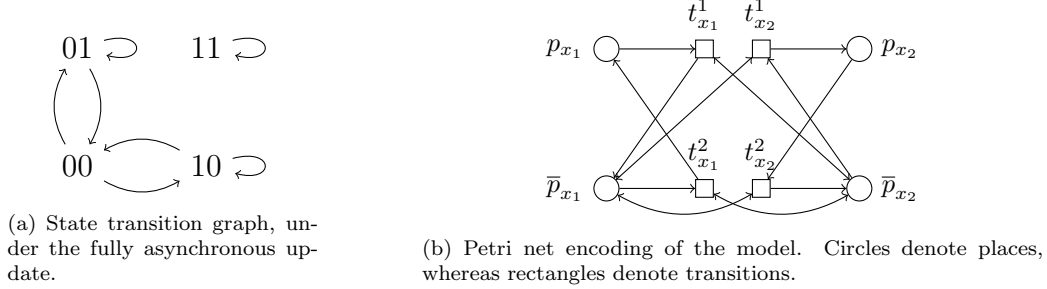


Figure 1: Dynamics and encoding of the Boolean network of Example 2.1.

2.3. Petri net encoding of Boolean networks

Definition 2.2. A Petri net is a weighted bipartite directed graph (P, T, W) , where P is a non-empty finite set of vertices called places, T is a non-empty finite set of vertices called transitions, $P \cap T = \emptyset$, and $W : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ is a weight function attached to the arcs.

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

The link between Boolean networks *à la* Thomas and Petri nets was originally established in [23] in order to make available formal methods like model-checking for the analysis of such systems. The basic encoding into 1-safe (i.e., never more than one token in each place) nets only holds for purely Boolean networks but was later extended to multivalued logical models in two ways, either in [24] with non 1-safe Petri nets or more recently in [22] with 1-safe nets but many more places.

187 Since our study is focused on Boolean networks, we briefly recall the orig-
188 inal encoding here. Its basis is that every node (*gene*) v of the original model
189 $\mathcal{N} = (V, F)$ is represented by two separate places (p_v and \bar{p}_v), corresponding
190 to its two states, active, and inactive, respectively. Each conjunct of the
191 logical function that activates the *gene* will lead to a transition t , consuming
192 the inactive place (i.e., a directional arc from \bar{p}_v to t), producing the active
193 place (i.e., a directional arc from t to p_v), and with all other literals both
194 consumed and produced (i.e., a bidirectional arc). And conversely for the
195 inactivation. Let s be a state of the Boolean network and m_s be its corre-
196 sponding marking in the encoded Petri net. It holds that $\forall v \in V, s(v) = 0$ if
197 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
198 at any marking m of the Petri net encoding a Boolean network, it always
199 holds that $m(p_v) + m(\bar{p}_v) = 1$.

200 The main property of this encoding is that it is completely faithful with
201 respect to the update scheme of the original Boolean network. For each node
202 v of \mathcal{N} , only transitions corresponding to v can change the current marking
203 of p_v or \bar{p}_v . In addition, at any marking at most one of such transitions is en-
204 abled because $m(p_v) + m(\bar{p}_v) = 1$ holds. Hence, for any update scheme in \mathcal{N} ,
205 we have a corresponding firing scheme in \mathcal{P} , which preserves the equivalence
206 between the dynamics of \mathcal{N} and \mathcal{P} [25].

207 For illustration, let us reconsider the Boolean network shown in Exam-
208 ple 2.1. Figure 1(b) shows the Petri net encoding of this Boolean network.
209 Place p_{x_1} (resp. \bar{p}_{x_1}) in \mathcal{P} represents the activation (resp. the inactivation) of
210 node x_1 in \mathcal{N} . Marking $\{p_{x_1}, \bar{p}_{x_2}\}$ in \mathcal{P} represents state 10 in \mathcal{N} . Transitions
211 $t_{x_1}^1$ and $t_{x_1}^2$ represent the update of node x_1 . Of course, in any marking $t_{x_1}^1$
212 and $t_{x_1}^2$ cannot be both enabled. Then, the fully asynchronous update scheme
213 in \mathcal{N} corresponds to the classical firing scheme in \mathcal{P} where only one of the
214 enabled transitions for a given marking will be fired [12].

215 Note that given a Boolean network in the standard **SBML-Qual** format [26],
216 i.e., the package of SBML v3 [27] for such models, one can easily obtain its
217 Petri net encoding in the Petri Net Markup Language (PNML)² standard
218 using the **bioLQM**³ library. This piece of software extracted from **GINsim** [28]
219 and part of the **CoLoMoTo**⁴ [29] software suite allows for easy conversion

²<https://www.pnml.org/>

³<http://www.colomoto.org/biolqm/>

⁴<http://colomoto.org/>

between standard formats. It also accepts many other common formats for Boolean networks, notably the `.bnet` files of the BoolNet [30, 20] tools. The conversion is executed as follows:

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

Note that transforming a Boolean network defined by its functions into its Petri net encoding roughly relies on obtaining conditions for the activation and inactivation of the states. In [23] this took the form of the whole truth table of the Boolean functions, but as shown in Appendix 1 of [22] computing Disjunctive Normal Forms (DNF) of each Boolean function is enough. Though this might appear quite computationally intensive it is important to remark first that contrary to the prime-implicants case, there is no need to find *minimal* DNFs. One way to look at this is to consider that this amounts to a similar approach as that used in [8] but with the encoding of both activation and inhibition functions as DNFs in order to take into account possible non-local-monotonicity. This does not change the worst-case-complexity (obtaining a single DNF being exponential) but might matter a lot in practice. As such, we will explore how this transformation, here using BDDs in `bioLQM` or directly in our tool using the `pyeda`⁵ library, and the one based on the most-permissive semantics compare in the Section 6 on evaluation.

2.4. Siphons

Siphons are a static and classical property of Petri nets [11]. Note however that the use of siphons for the analysis of biological models, though it is not new, has been mostly relevant to the ODE-based continuous semantics of Chemical Reaction Networks [31, 32, 33]. We recall here the basic definition establishing that to produce something in a siphon you must consume something from the siphon. This corresponds to the idea that a siphon is a set of places that once unmarked remains unmarked.

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

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

Note that \emptyset is trivially a siphon.

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 conventionally $\text{pred}(S) = \text{succ}(S) = \emptyset$. We have an important property on siphons [34] as follows.

⁵<https://pyeda.readthedocs.io/en/latest/>

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

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

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

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

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

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

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

and

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

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

277 *Proof.* First, we show that if m is a trap space of \mathcal{N} , then S is a conflict-free
 278 siphon of \mathcal{P} (*). If $D_m = \emptyset$, then $S = \emptyset$ is trivially a conflict-free siphon of
 279 \mathcal{P} . Thus, we consider the case that $D_m \neq \emptyset$ (resp. $S \neq \emptyset$). Assume that S
 280 is not a siphon of \mathcal{P} . Then, there is a transition $t \in T$ such that $S \cap \text{succ}(t) \neq \emptyset$
 281 but $S \cap \text{pred}(t) = \emptyset$. This implies that there is a place $p \in S$ such that
 282 $p \in \text{succ}(t)$ but $p \notin \text{pred}(t)$. Let v be the node in \mathcal{N} corresponding to p . By
 283 the characteristics of the encoding [23], there is a directional arc from t to p
 284 and a directional arc from the complementary place of p to t . Without loss
 285 of generality, we assume that $p = p_v$, then there is a directional arc from t
 286 to p_v and a directional arc from \bar{p}_v to t . We follow the following procedure
 287 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
 288 the corresponding marking in \mathcal{P} of s . For every place $p' \in \text{pred}(t)$, let p'' be
 289 the complementary place of p' and v' be the corresponding node in \mathcal{N} of p'
 290 and p'' . If $p'' \notin S$, then $v' \notin D_m$ and we can always set a Boolean value to
 291 $s(v')$ 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
 292 set $s(v') = m(v')$. In this case, if $p' = p_v$ then $s(v') = m(v') = 1$ leading to
 293 $m_s(p') = 1$, if $p' = \bar{p}_v$ then $s(v') = m(v') = 0$ leading to $m_s(p') = 1$. For
 294 the remaining nodes of \mathcal{N} , we can always set Boolean values to these nodes
 295 to preserve that $s \in \mathcal{S}_{\mathcal{N}}[m]$. We also have $m_s(p_v) = 0$ by the characteristics
 296 of the encoding [23]. Now, t is enabled at marking m_s . Its firing leads to
 297 a new marking m'_s such that $m'_s(p_v) = 1$ and $m'_s(\bar{p}_v) = 0$. Let s' be the
 298 corresponding state in \mathcal{N} of m'_s . We have $s'(v) = 1$ because $m'_s(p_v) = 1$ and
 299 $m(v) = 0$ because $p_v \in S$. This implies that $s' \notin \mathcal{S}_{\mathcal{N}}[m]$. For any firing
 300 scheme of \mathcal{P} , the firing of t always happens. Since a firing scheme of \mathcal{P}
 301 is equivalent to an update scheme of \mathcal{N} , s can escape from the trap space m
 302 for any update scheme of \mathcal{N} , which contradicts to the property of a trap
 303 space. Hence, S is a siphon of \mathcal{P} . By the definition of a mirror, S is also a
 304 conflict-free one.

305 Second, we show that if S is a conflict-free siphon of \mathcal{P} , then m is a trap
 306 space of \mathcal{N} (**). By the definition of a mirror, m is a subspace of \mathcal{N} . Let
 307 s be an arbitrary state in $\mathcal{S}_{\mathcal{N}}[m]$ and m_s be its corresponding marking in
 308 \mathcal{P} . Assume that there is a place $p \in S$ such that $m_s(p) = 1$. Let v be the
 309 corresponding node in \mathcal{N} of p . Since $p \in S$, $v \in D_m$ and $m(v) = s(v)$. If
 310 $p = p_v$, then $m_s(p_v) = 1$ leading to $m(v) = s(v) = 1$ by the characteristics of
 311 the encoding [23]. By the definition of a mirror, $m(v) = 0$ because $p_v \in S$,
 312 which is a contradiction. It is symmetric for the case that $p = \bar{p}_v$. Hence,
 313 $m_s(p) = 0, \forall p \in S$. In any marking m'_s reachable from m_s regardless of the
 314 firing scheme of \mathcal{P} , we have $m'_s(p) = 0, \forall p \in S$ by the dynamical property on

315 markings of a siphon [34]. Let s' be the corresponding state in \mathcal{N} of m'_s . For
 316 every node $v \in D_m$, we have all two cases as follows. Case 1: $p_v \in S$, then
 317 $m'_s(p_v) = 0$, thus $s'(v) = 0 = m(v)$. Case 2: $\bar{p}_v \in S$, then $m'_s(\bar{p}_v) = 0$, thus
 318 $s'(v) = 1 = m(v)$. Hence, $s'(v) = m(v)$ for every $v \in D_m$. Then, $s' \in \mathcal{S}_{\mathcal{N}}[m]$.
 319 By the definition of a trap space and the arbitrariness of s , m is a trap space
 320 of \mathcal{N} .

321 From (*) and (**), we can conclude the proof. \square

322 From the proof of Theorem 3.1, we can see that the theorem holds for
 323 any update scheme associated to the Boolean network. Since the Petri net
 324 encoding of a Boolean network is independent of its update scheme and
 325 siphons are a static property of a Petri net, we can imply that trap spaces of a
 326 Boolean network are independent of its update scheme. Note that the original
 327 proof for this property of trap spaces (see Theorem 1 of [7]) only considers
 328 the two popular update schemes (i.e., synchronous and fully asynchronous).
 329 Theorem 3.1 exhibits the very first theoretical application of the connection
 330 between trap spaces of Boolean networks and siphons of Petri nets.

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

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

342 Second, we show that if S is a maximal conflict-free siphon of \mathcal{P} , then
 343 m is a minimal trap space of \mathcal{N} (**). Since S is a conflict-free siphon of \mathcal{P} ,
 344 m is a trap space of \mathcal{N} by Theorem 3.1. Assume that m is not minimal.
 345 Then, there is another trap space m' such that $m' < m$. By the definition of
 346 the partial order $<$ on subspaces, $\mathcal{S}_{\mathcal{N}}[m'] \subset \mathcal{S}_{\mathcal{N}}[m]$. Let S' be the mirror of
 347 m' . S' is a conflict-free siphon by Theorem 3.1. Following the definition of
 348 a mirror, $S \subset S'$, which contradicts to the maximality of S . Hence, m is a
 349 minimal trap space of \mathcal{N} .

350 From (*) and (**), we can conclude the proof. \square

351 We here showcase a theoretical application of the connection between
 352 trap spaces in Boolean networks and conflict-free siphons in Petri nets. We
 353 use it to prove a property of minimal trap spaces, which has surprisingly
 354 not been formally proved. Specifically, all minimal trap spaces of a Boolean
 355 network are mutually disjoint. This property is important because we can
 356 use it to approximate the set of attractors of the Boolean network under
 357 any update scheme [7] or to compute exactly the set of complex attractors
 358 of the Boolean network under the fully asynchronous update scheme [35].
 359 Note that it would be not difficult to obtain a direct proof on trap spaces
 360 for this property, which follows the same structure as the proof on siphons.
 361 However, we emphasize here the potential of using the connection between
 362 Boolean networks and Petri nets to explore and prove properties of trap
 363 spaces in Boolean networks.

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

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

371 Consider two any distinct minimal trap spaces m_1 and m_2 . Assume that
 372 $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] \neq \emptyset$. Let S_1 and S_2 be the mirrors of m_1 and m_2 , re-
 373 spectively. By Theorem 3.2, S_1 and S_2 are maximal conflict-free siphons
 374 of \mathcal{P} . We have that $S = S_1 \cup S_2$ is also a siphon because of Proposi-
 375 tion 2.1. For every node $v \in V$, assume that $p_v \in S$ and $\bar{p}_v \in S$ hold.
 376 Since S_1 and S_2 are conflict-free, there are all two cases. Case 1: $p_v \in S_1$
 377 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
 378 $m_1(v) \neq m_2(v)$, $m_1(v) \neq \star$, $m_2(v) \neq \star$, then $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$. This is a
 379 contradiction. Hence, for every node $v \in V$, $p_v \in S$ and $\bar{p}_v \in S$ cannot hold
 380 together. Therefore, S is conflict-free. Now, we have that S is a conflict-free
 381 siphon but $S_1 \subset S$ or $S_2 \subset S$ holds because $S_1 \neq S_2$. This contradicts to the
 382 maximality of S_1 and S_2 . Hence, $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$ holds.

383 □

384 A natural computational application of Theorem 3.1 is that we can effi-
 385 ciently decide whether a subspace m is a trap space. In `PyBoolNet` [20], this
 386 is checked by using the percolation on the prime-implicants of the Boolean

387 functions. As we have mentioned at the beginning of this article, the compu-
388 tation of prime-implicants is a demanding task for complex Boolean networks,
389 even is sometimes intractable. Hence, the checking method in [20] shows its
390 limitations. Instead, we can first compute the mirror S_m of m in the Petri
391 net encoding. Then, by Proposition 2.1 and Theorem 3.1, we can check if
392 $\text{pred}(S_m) \subseteq \text{succ}(S_m)$. Note that the Petri net construction is less com-
393 putationally demanding than the prime-implicant computation because it
394 only requires computing generic (not prime) implicants of the Boolean func-
395 tions [22]. In addition, the time complexity of the above checking method is
396 quadratic in the number of transitions of the Petri net in worst cases.

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

406 Note that there are no existing methods specifically designed for comput-
407 ing maximal conflict-free siphons (even maximal siphons) of a Petri net. The
408 reason might be that researchers mainly focus on minimal generic siphons [34]
409 in the field of Petri nets. Hence, we here propose several methods for comput-
410 ing maximal conflict-free siphons of a Petri net. The details of the proposed
411 methods shall be given in the next section.

412 4. Computation methods

413 4.1. Characterization

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

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

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

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

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

416 4.2. Constraint satisfaction problem

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

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

- 421 • $R = P$, i.e., a variable is introduced for each place of \mathcal{P} ,
- 422 • $D(p) = \mathbb{B}$ for all $p \in R$, i.e., the variables are Boolean,
- 423 • $C = \{\neg p_v \vee \neg \bar{p}_v = 1 \mid \forall v \in V\} \wedge \{(p = 1 \rightarrow \bigvee_{p' \in \text{pred}(t)} p' = 1) \mid p \in$
424 $P, t \in \text{pred}(p)\}$.

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

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

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

426 *Proof.* By the former part $\neg p_v \vee \neg \bar{p}_v = 1$ of C , the conflict-freeness is imposed
427 because for any satisfiable valuation r , $r(p_v) = r(\bar{p}_v) = 1$ is impossible for all
428 $v \in V$. As shown in [17], the latter part of C can characterize the set of all
429 generic siphons of \mathcal{P} . Hence, we can conclude the proof.

430 □

431 In [17], the set of all siphons of a given Petri net is characterized by a sim-
432 ilar Boolean CSP except the conflict-freeness constraint. From the encoded
433 CSP, the set of all *minimal* siphons of the Petri net can be enumerated in the
434 set inclusion order. For enumerating siphons in the set inclusion order, the
435 proposed method by [17] uses the technique that labels directly the Boolean

436 variables with increasing value selection (i.e., to test first the absence, then
 437 the presence of a place in the candidate solution). The method has two
 438 implementations, one uses an iterated SAT procedure and the other uses
 439 Constraint Programming (CP) with backtracking.

440 One natural question is that how to use the CSP-based method for enu-
 441 merating all the maximal conflict-free siphons of a Petri net encoding a
 442 Boolean network? Of course, the set of all conflict-free siphons of the Petri
 443 net can easily be characterized by the CSP model presented in [17] along with
 444 the additional constraint $\neg p_v \vee \neg \bar{p}_v = 1$, for each $v \in V$, which represents
 445 the conflict-freeness. However, the main concern is to enumerate all the
 446 *maximal* ones, which is not trivial to adapt from the CSP-based method.
 447 By Proposition 4.1, the set of all maximal conflict-free siphons of \mathcal{P} can be
 448 enumerated in the (maximality) set inclusion order, by restarting the search
 449 each time a conflict-free siphon S is found, with the following additional con-
 450 straint for disallowing any subset of that conflict-free siphon: $\bigvee_{p \notin S} p = 1$.
 451 For enumerating conflict-free siphons in the set inclusion order, we can use
 452 the same technique as used in [17] but with the opposite setting, i.e., labeling
 453 directly the Boolean variables with decreasing value selection. The correct-
 454 ness of this technique comes from the fact that once S is found, it is the
 455 conflict-free siphon of maximum cardinality among all the remaining feasible
 456 conflict-free siphons. Similar to [17], the newly CSP-based method can also
 457 be implemented with SAT and CP solvers.

458 This method was implemented using the state-of-the-art CP solver Chuffed⁶ [36]
 459 via its MiniZinc [37] interface. Because it is a high-level interface, the
 460 backtrack-and-replay method of [17] was not used but rather the alterna-
 461 tive implementation with two global constraints for lexicographic ordering
 462 (ensuring enumeration of solutions) and iterated non-subset of each already
 463 found solution (for maximality).

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

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

⁶<https://github.com/chuffed/chuffed>

464 We set a soft clause for each variable of the CSP and then use a “minimal
 465 correction subset” blocking strategy, which will ensure set-inclusion maxi-
 466 mality of the solutions. This is what is implemented in **Trappist** using the
 467 RC2 MaxSAT solver [38] available through the **python-sat** package⁷.

468 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

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

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 set of atoms representing places in $\text{pred}(t)$. The rule (2) is translated into the ASP rule

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

for each $v \in V$. This ASP rule guarantees that two places representing the same node in \mathcal{N} never belong to the same siphon of \mathcal{P} , representing the conflict-freeness. Naturally, a Herbrand model (see, e.g., [39]) of \mathcal{L} is equivalent to a conflict-free siphon of \mathcal{P} . To guarantee that a Herbrand model is also a stable model (an answer set), we need to add to \mathcal{L} the two choice rules

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

469 for each $v \in V$. Note that the number of atoms of \mathcal{L} is only $2n$, whereas
 470 the ASP encoding shown in [7] has as many atoms as the number of prime-
 471 implicants of the Boolean network and that number might be exponential in
 472 n . In [8], there is an ASP characterization of trap spaces that does not rely
 473 on minimal DNFs either and thus seems very similar to our ASP encoding.
 474 Remarkably it only requires the DNF for the *activation* part, using the in-
 475 formation that it will only be used for locally-monotonic Boolean networks.
 476 We would therefore expect that, when available, it will have comparable per-
 477 formance on the ASP part (the ASP program would be approximately twice
 478 smaller, though redundancy is not always bad in that field), but can also

⁷<https://pysathq.github.io/docs/html/api/examples/rc2.html>

479 avoid combinatorial explosion of the Petri net encoding for some formula
 480 where the activation DNF is simple but the inhibition is not. Since `mpbn` is
 481 included in our benchmark this will be evaluated in our experiments.

482 Now, a solution (simply an answer set) $A \subseteq \mathcal{A}$ of \mathcal{L} is equivalent to a
 483 conflict-free siphon S of \mathcal{P} , thus a trap space m of \mathcal{N} . The conversion from A
 484 to m is straightforward. If $\mathbf{p-v} \in A$ then $v \in D_m$ and $m(v) = 0$. Conversely,
 485 if $\mathbf{n-v} \in A$ then $v \in D_m$ and $m(v) = 1$. Otherwise, $v \notin D_m$. Comput-
 486 ing multiple answer sets is built into ASP solvers and the solving collection
 487 `POTASSCO` [39] also features the option to find set-inclusion maximal answer
 488 sets with respect to the set of atoms. Naturally, a set-inclusion maximal
 489 answer set of \mathcal{L} is equivalent to a maximal conflict-free siphon of \mathcal{P} , thus a
 490 minimal trap space of \mathcal{N} . By using this built-in option, we can compute all
 491 the set-inclusion maximal answer sets of \mathcal{L} (resp. all the minimal trap spaces
 492 of \mathcal{N}) in one execution.

493 4.4. Integer linear programming-based method

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

$$\mathbf{a} \leq \mathbf{a}_1 + \dots + \mathbf{a}_k$$

where \mathbf{a} is the binary variable representing place p and $\{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ is the
 set of binary variable representing places in $\text{pred}(t)$. The rule (2) is translated
 into the ILP inequality

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

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

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

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

495 We can see the similarity between \mathcal{I} and the encoded ASP shown in the
 496 previous subsection. However, due to the nature of solutions of an ILP, it is

497 hard to compute all the set-inclusion maximal solutions of \mathcal{I} in one execution
 498 of an ILP solver. Hence, we propose an iterative approach as follows.

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

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

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

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

499 where $\mathbf{a-p}$ is the binary variable presenting place p . Now, we solve \mathcal{I} again to
 500 find a new solution. If a new solution I' exists, then let S' be its corresponding
 501 conflict-free siphon. Indeed, abide by the newly added inequality, we have
 502 $S' \cap (P \setminus S^*) \neq \emptyset$ because there is some $\mathbf{a-p}$ with $p \in P \setminus S^*$ such that
 503 $I'(\mathbf{a-p}) = 1$. This implies that it is impossible that $S' = S^*$ or $S' \subset S^*$.
 504 By the objective function, it means that S' is the conflict-free siphon of
 505 maximum cardinality among the conflict-free siphons that are not contained
 506 in S^* . Hence, S' is also a maximal conflict-free siphon. Again, we add to \mathcal{I}
 507 a new inequality with respect to the newly found siphon. The above process
 508 is iterated until \mathcal{I} becomes unfeasible, this means that there is no further
 509 maximal conflict-free siphon. Thus, all the maximal conflict-free siphons of
 510 the Petri net have been found.

511 Since we used the MiniZinc framework to interface with the CP solver,
 512 it was simple to make the slight modifications described above and use that
 513 same interface to call the Coin-OR CBC solver⁸ [40].

514 4.5. Computation of special types of trap spaces

515 In the field of systems biology, biologists may want to compute more
 516 special types of trap spaces beyond minimal trap spaces [20], which also play
 517 crucial roles in analysis and control of Boolean networks [21, 19]. We shall
 518 show that our proposed methods can be easily adjusted to compute such

⁸<https://github.com/coin-or/Cbc>

519 popular types of trap spaces. We illustrate the adjustments via the ASP-
 520 based method (see Subsection 4.3) because ASP is declarative by nature,
 521 but these adjustments are completely applicable for other approaches such
 522 as MaxSAT, CP, and ILP.

523 First, the work by [19] uses the concept of *stable motifs* to build the suc-
 524 cession diagram of a Boolean network, a summary of the decisions in the
 525 network dynamics that lead to successively more restrictive nested stable
 526 motifs. The succession diagram is useful for control and decision making
 527 on this Boolean network. In particular, the proposed control methods are
 528 independent to the update scheme. It has been shown that a stable motif of
 529 a Boolean network is equivalent to a maximal trap space of this Boolean net-
 530 work [19]. Hence, it is necessary to develop an efficient method for computing
 531 maximal trap spaces of a Boolean network. We shall show how to adjust the
 532 ASP-method presented in Subsection 4.3 to compute maximal trap spaces.

We first provide the definition of maximal trap spaces. Let ε be the special
 trap space of \mathcal{N} where all the nodes are free. Of course, ε corresponds to the
 special conflict-free siphon \emptyset . A trap space m is called maximal if $m \neq \varepsilon$ and
 there is no other trap space m' such that $m' \neq \varepsilon$ and $m < m'$. Analogously,
 a conflict-free siphon S is called minimal if $S \neq \emptyset$ and there is no other
 trap space S' such that $S' \neq \emptyset$ and $S' \subset S$. By using the reasoning similar
 to the proof of Theorem 3.2, we can easily conclude that a maximal trap
 space of \mathcal{N} is equivalent to a minimal conflict-free siphon of its encoded
 Petri net \mathcal{P} . Let \mathcal{L} be the ASP characterizing all conflict-free siphons of \mathcal{P}
 (see Subsection 4.3). Naturally, we need to exclude \emptyset from the solution space
 of \mathcal{L} (equivalently exclude ε from the set of trap spaces). To do this, we add
 to \mathcal{L} the ASP rule

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

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

Second, we consider *fixed points* in Boolean networks. To date, the anal-
 ysis of the fixed points of a Boolean network remains a very useful tool in
 understanding the behavior of complex biological models not only due to the
 fact that in some cases the full computation of complex attractors remains
 intractable, but also because for many biological systems, the expected long-
 term behavior is not cyclic [41]. Furthermore, the fixed point computation is

also the crucial starting point for several state-of-the-art methods for computing complex attractors of Boolean networks [35]. Let s be a fixed point of a Boolean network \mathcal{N} . We have a subspace m corresponding to s as follows: $\forall v \in V, m(v) = s(v)$, i.e., all nodes are fixed in m . Clearly, s is a trap set of \mathcal{N} regardless of the update scheme. Hence, m is a trap space of \mathcal{N} . In addition, since $|S_{\mathcal{N}}[m]| = 1$, m is also a minimal trap space. To compute all fixed points of \mathcal{N} , we can add more constraints to the encoded ASP characterizing all conflict-free siphons (equivalently trap spaces). For every $v \in V$, we add to the encoded ASP the rule

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

536 that ensures that for every conflict-free siphon S , it contains either $\mathbf{p-v}$ or $\mathbf{n-v}$
 537 for every $v \in V$. Equivalently, the trap space corresponding to S is always
 538 a fixed point. Now, the set of answer sets of the encoded ASP is equivalent
 539 to the set of fixed points of \mathcal{N} . In particular, when solving the encoded ASP
 540 using an ASP solver, we do not need to use the built-in option for computing
 541 set-inclusion maximal answer sets. Note that we can also build another ASP
 542 characterizing all fixed points of \mathcal{N} based on the equivalence between a fixed
 543 point of \mathcal{N} and a deadlock of its Petri net encoding [22]. This approach may
 544 give a more compact ASP.

Third, we consider the trap spaces *intersecting* a given subspace m^* of a Boolean network. Such trap spaces are used in the trap space-based control method [21]. A trap space m intersects m^* if and only if $S_{\mathcal{N}}[m] \cap S_{\mathcal{N}}[m^*] \neq \emptyset$. It follows that for every v , if $m^*(v) = 0$ then $m(v) = 0$ or $m(v) = \star$, if $m^*(v) = 1$ then $m(v) = 1$ or $m(v) = \star$. For the former case, we add to \mathcal{L} the ASP rule

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

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

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

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

Finally, we consider the trap spaces that are *inside* a given subspace m^* of a Boolean network. Such trap spaces are used in the iterative procedure of building the succession diagram of a Boolean network [19], which is hierarchical. We first adjust \mathcal{L} to characterize all such trap spaces. A trap space

m is inside m^* if and only if $m(v) = m^*(v)$ for every $v \in D_{m^*}$. If $m^*(v) = 0$, we add to \mathcal{L} the ASP rule

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

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

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

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

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

547 where $\{v_{i_1}, \dots, v_{i_k}\}$ is the set of free nodes of m^* . This rule ensures that
 548 $m \neq m^*$. In the case that $m^* = \varepsilon$, we have all maximal trap spaces of the
 549 original Boolean network.

550 5. Motivating example

551 For a few years now we have been collaborating with biologists who build
 552 very large detailed and annotated maps and now wish to analyze the dy-
 553 namics of the corresponding models. One of the main maps studied this way
 554 represents knowledge about the Rheumatoid Arthritis [42], and was the main
 555 motivation for the development of a tool to automatically transform it into
 556 an executable Boolean network [6]. In the supplementary material of the pa-
 557 per, an excerpt of the map, focused around the apoptosis (cell death) module
 558 is transformed into a model of *reasonable* size, namely 180 Boolean variables
 559 (model `F5_RA_apoptosis_executable_module.sbml` of supplementary ma-
 560 terial S3, and model “RA-apoptosis” of Section 6). The study of such model,
 561 though, is a big hurdle. Indeed, as stated in the article about another model
 562 of the same size: “*The size of the CaSQ-inferred MAPK model (181 nodes)*
 563 *made the calculation of stable states a non-realistic endeavour.*”

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

571 However, using a state-of-the-art tool like `PyBoolNet` [7] on that model
572 actually fails at the phase of prime-implicant generation. `mpbn` [9] can return
573 the first 1000 solution within 1.43s, but indeed, it limits the modeling range
574 of the modelers as it does not permit using non-locally-monotonic Boolean
575 functions. This is also true for the Alzheimer model also mentioned in that
576 same article and originally from [43] (F4 file in the original supplementary
577 material, and “Alzheimer” in Table 2), where `PyBoolNet` also fails at the
578 prime-implicant computation and `mpbn` does not give any answer because
579 this model is actually non-locally-monotonic. The current practice usually
580 revolves then around fixing some source nodes to plausible values and re-
581 ducing the model accordingly. While this approach makes sense, it relies
582 on potentially arbitrary decisions, and *hides away* critical modelling choices
583 that were actually not part of the original Boolean network or even of the
584 starting map.

585 For the “RA-apoptosis” model, using the ASP-based method presented
586 in Section 4.3, it is possible to obtain the first 1000 minimal trap spaces
587 (including ones that contain more than one state) within 0.19s, which is
588 much quicker than `mpbn`. The needed time for the “Alzheimer” model is
589 0.79s. Unfortunately since this method was not available at the time, the
590 analysis of the model remained very high-level and qualitative, instead of
591 being able to use the rich information of computed minimal trap spaces.

592 6. Evaluation

593 To evaluate the performance of the newly proposed methods (imple-
594 mented as a Python package named `Trappist` and available on the Python
595 package index⁹) and the state-of-the-art methods (`bioLQM`¹⁰, `PyBoolNet` [7,
596 20], and `mpbn` [9]), we compared them on both `PyBoolNet`’s own model repos-
597 itory and many real-world models from various sources in the literature. To
598 our knowledge, these models are a highly representative sample of Boolean
599 models currently available. It is worth noting that `mpbn` [9] only handles
600 locally-monotonic models, whereas the other methods can handle general
601 models. To obtain a more comprehensive comparison, we also used random
602 models generated by a third-party software `BoolNet R` package [30]. As ex-
603 plained in Section 5, in our benchmarks, we only searched for the first 1000

⁹<https://pypi.org/project/trappist/>

¹⁰<http://colomoto.org/biolqm/doc/tools-trap-space.html>

minimal trap spaces for each model. It is worth noting that unlike existing analysis shown in the literature, we did not fix specific values for source nodes in all the considered models.

To solve the ASP problems, we used the same ASP solver `Clingo` [39] and the same configuration as that used in `PyBoolNet` [7, 20] 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 some Jupyter notebooks realizing the benchmarks (and named `TCS-Benchmark-<...>.ipynb`) 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` repository¹¹. 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 minimal trap spaces. “DNF” means that the method did not finish the computation within the time limit of three minutes. In the case of `bioLQM`, “N/A” means that the number of all minimal trap spaces of the model is larger than 1000 and we did not recorded the running time of `bioLQM` because it always requires to compute all minimal trap spaces. A number in bold indicates a ratio greater than three compared to the best result. “NM” indicates a non-locally-monotonic model. There are four variants of `Trappist: SAT` (i.e., `Trappist-MaxSAT`, the MaxSAT-based method shown in Subsection 4.2), `CP` (i.e., `Trappist-CP`, the CP-based method shown in Subsection 4.2), `ILP` (i.e., `Trappist-ILP`, the ILP-based method shown in Subsection 4.4), and `ASP` (i.e., `Trappist-ASP`, the ASP-based method shown in Subsection 4.3).

We first analyze the results of the four variants of `Trappist`. We can see that `Trappist-MaxSAT` and `Trappist-ASP` are comparable in most mod-

¹¹<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	CP	ILP	ASP
1 arellano_rootstem	9	4	0.13	0.01	0.00	0.00	0.97	0.96	0.01
2 calzone_cellfate	28	27	0.12	0.02	0.01	0.01	5.59	6.03	0.01
3 dahlhaus_neuroplastoma	23	32	0.11	0.03	0.01	0.01	6.56	6.99	0.01
4 davidich_yeast	10	12	0.11	0.02	0.01	0.01	2.56	2.21	0.01
5 dinwoodie_life	15	7	0.11	0.01	0.00	0.01	1.68	1.39	0.01
6 dinwoodie_stomatal	13	1	0.10	0.01	0.00	0.00	0.39	0.29	0.01
7 faure_cellcycle	10	2	0.11	0.02	0.01	0.01	0.58	0.46	0.01
8 grieco_mapk	53	18	0.19	0.03	0.02	0.03	3.93	10.46	0.02
9 irons_yeast	18	1	0.12	0.03	0.01	0.01	0.37	0.39	0.02
10 jaoude_thdiff	103	1000 ⁺	N/A	0.85	0.45	0.56	DNF	DNF	0.09
11 klamt_tcr	40	8	0.11	0.01	0.01	0.01	1.98	1.22	0.02
12 krumsiek_myeloid	11	6	0.10	0.01	0.00	0.00	1.48	1.26	0.01
13 multivalued	13	4	0.10	0.01	0.00	0.00	0.93	0.86	0.01
14 n12c5	11	5	0.11	17.83	0.01	0.01	1.21	1.10	0.01
15 n3s1c1a	2	2	0.10	0.01	0.00	0.00	0.63	0.49	0.01
16 n3s1c1b	2	2	0.09	0.02	0.00	0.00	0.56	0.49	0.01
17 n5s3	4	3	0.10	0.02	NM	0.00	0.74	0.69	0.01
18 n6s1c2	5	3	0.10	0.02	0.00	0.00	0.91	0.59	0.01
19 n7s3	6	3	0.11	0.02	0.00	0.00	0.79	0.68	0.01
20 raf	3	2	0.10	0.01	0.00	0.00	0.55	0.39	0.01
21 randomnet_n15k3	15	3	0.10	0.02	NM	0.01	0.77	0.67	0.01
22 randomnet_n7k3	7	10	0.10	0.01	NM	0.00	2.07	1.46	0.01
23 remy_tumorigenesis	34	25	0.15	0.94	0.02	0.02	5.98	7.98	0.02
24 saadatpour_guardcell	13	1	0.10	0.06	0.00	0.00	0.53	0.45	0.02
25 selvaggio_emt	56	1000 ⁺	N/A	0.48	0.28	0.28	DNF	DNF	0.09
26 tournier_apoptosis	12	3	0.10	0.01	0.00	0.00	0.74	0.75	0.01
27 xiao_wnt5a	7	4	0.10	0.01	0.00	0.00	1.00	0.89	0.01
28 zhang_tlgl	60	156	0.60	0.09	0.09	0.07	37.26	DNF	0.04
29 zhang_tlgl_v2	60	258	0.64	0.04	0.08	0.11	69.95	DNF	0.04

els, but Trappist-ASP is much faster for the jaoude_thdiff and selvaggio_emt models where the number of minimal trap spaces is greater than 1000. The latter can be explained by the fact that Trappist-MaxSAT follows an iterative approach, i.e., it restarts the search with a new constraint each time a solution is found (see Subsection 4.2). This iterative approach may be less efficient than the way ASP solvers use to enumerate multiple solutions

(answer sets), which is an advantage of ASP solvers [39]. Hence, when the number of solutions increases, the inferiority of **Trappist-*MaxSAT*** compared to **Trappist-*ASP*** will be exhibited more clearly. The two remaining variants, **Trappist-*CP*** and **Trappist-*ILP***, are much less efficient than **Trappist-*MaxSAT*** and **Trappist-*ASP*** in every model, even are more than three orders of magnitude slower in some models. The first reason for their bad performance is that they are also iterative methods like **Trappist-*MaxSAT***, thus they are not efficient for "enumeration" problems. Upon closer inspection, for the Boolean CSP characterizing conflict-free siphons, **CP** seems to be something that is a "less-efficient-SAT", handling mostly Boolean constraints and making little use of the global constraints only added for the iterative part. For **ILP**, it may be even worse, since the problem is purely Boolean (no real or integer numbers whatsoever). This is confirmed by the observation that for some quite large models (e.g., the *grieco_mapk*, *zhang_tlg1*, and *zhang_tlg1.v2* models), **Trappist-*ILP*** is much slower than **Trappist-*CP***. Note that the inferiority of **ILP** compared to **ASP** with respect to the trap space enumeration has been reported in [7]. Hereafter, we shall compare the best variant of **Trappist** (i.e., **Trappist-*ASP***) with other methods.

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. However upon closer inspection, we can see some notable differences. First, **Trappist-*ASP*** is far more efficient than **bioLQM** in every model with speedups between $5\times$ and $16\times$. Second, for small models, **PyBoolNet** and **mpbn** are comparable to **Trappist-*ASP***. However, on every model that was a bit challenging for **PyBoolNet** or **mpbn**, **Trappist-*ASP*** is far more efficient with speedups between $3\times$ and $5\times$ for the case of **mpbn**, and between $5\times$ and $1783\times$ for the case of **PyBoolNet**. In particular, the second best variant of **Trappist** (i.e., **Trappist-*MaxSAT***) is even far more efficient than **bioLQM** and **PyBoolNet**, and is comparable to **mpbn** on every model. It is worth noting that for 3 of the 29 models, **mpbn** did not give any answer because these models are locally-monotonic but all the other methods did, which confirms the limit of **mpbn** on the applicable class of models.

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. **BBM** consists of 211 models (24

681 out of them are non-locally-monotonic), peaking at 321 nodes, 1100 regula-
682 tions among the nodes, and 133 source nodes, respectively. It is released and
683 maintained at <https://github.com/sybila/biodivine-boolean-models>.
684 We here tested all the compared methods on this model repository.

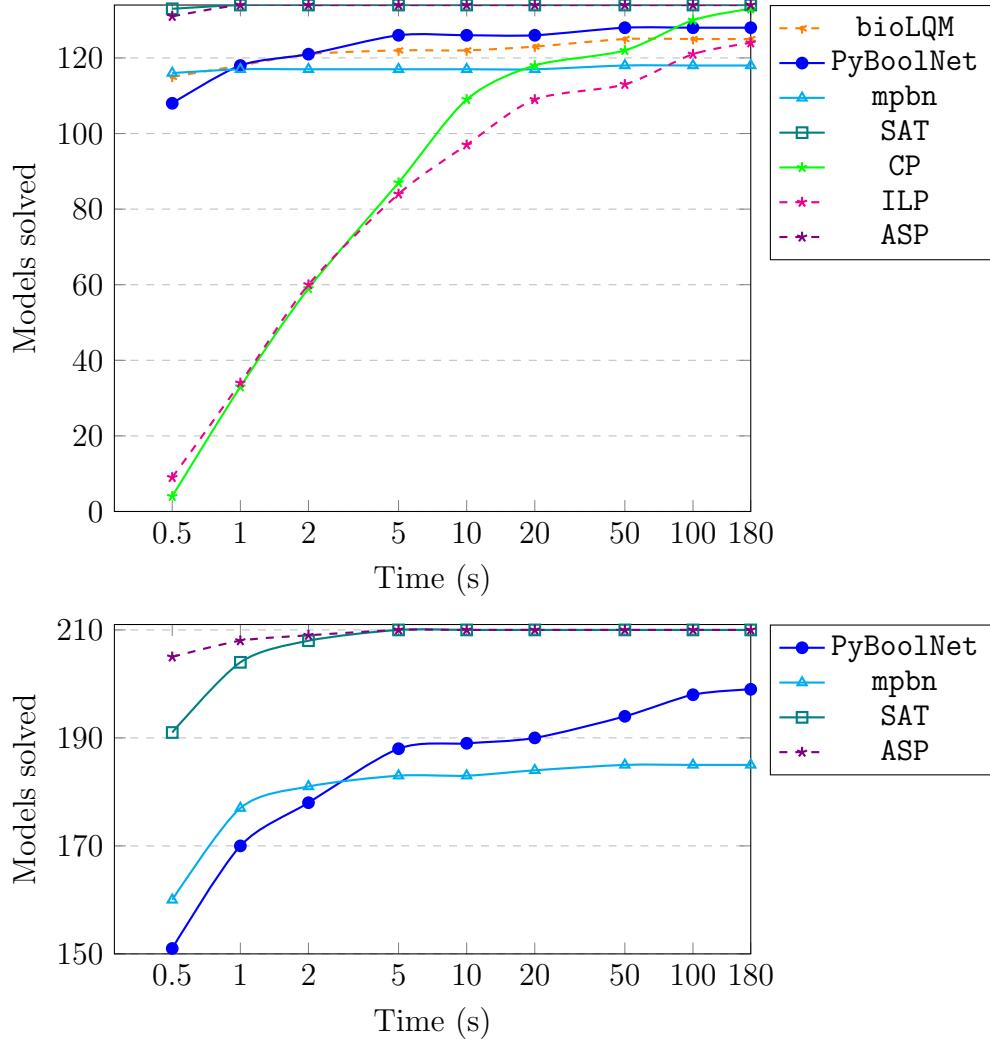


Figure 2: Cumulative numbers of the BBM models that have less than 1000 minimal trap spaces (upper panel) and BBM models solved by the compared methods with respect to enumerating the first 1000 minimal trap spaces (lower panel).

685 Figure 2 (upper panel) shows cumulative numbers of the BBM models that
686 have less than 1000 minimal trap spaces solved by the compared methods

with respect to enumerating the first 1000 minimal trap spaces. The number of such models is 134 (per all 211 models), and 15 of them are non-locally-monotonic. This model set allows us to fairly consider **bioLQM** for comparison, since **bioLQM** always requires to compute all minimal trap spaces. We can first see that **Trappist-ASP** and **Trappist-MaxSAT** are still the two best methods as they can handle every model within 1s as well as they always can handle more models than all the remaining methods on every time limit. Second, **Trappist-CP** is better than **Trappist-ILP**, which is consistent with their comparison shown in the previous subsection. Third, one notable remark is that for the time limit of 100s or 180s, **Trappist-CP** can handle more models than all **bioLQM**, **PyBoolNet**, and **mpbn**. This remark shows that even with a not best implementation, our alternative approach is still better than the state-of-the-art methods on a certain set of real-world models. This is supported by the fact that our alternative approach avoids the need for computing prime implicants (as opposed to **PyBoolNet**) and can handle non-locally-monotonic Boolean networks (as opposed to **mpbn**).

Figure 2 (lower panel) shows cumulative numbers of the **BBM** models solved by the compared methods (except **bioLQM**, **Trappist-CP**, and **Trappist-ILP**) with respect to enumerating the first 1000 minimal trap spaces. We omit the results of **Trappist-CP** and **Trappist-ILP** because they can handle no model with more than 1000 minimal trap spaces. Again, we can see that **Trappist-ASP** and **Trappist-MaxSAT** are the two best methods as they can handle every but one model within 5s. They also always handle many more models than both **PyBoolNet** and **mpbn** on every time limit. Note that with the time limit of 0.5s, **Trappist-ASP** can handle 14 more models than **Trappist-MaxSAT**, which is opposed to the case of models with less than 1000 minimal trap spaces (see Figure 2 (upper panel)). This observation confirms the disadvantage of **Trappist-MaxSAT** compared to **Trappist-ASP** for the case of many minimal trap spaces.

6.3. Selected models

We used a set of real-world Boolean networks lying in various scales collected from numerous bibliographic sources in the literature. Most of these models are quite big (in size), complex (i.e., having high average in-degree, which is related to the number of prime-implicants), and have never been fully analyzed. Note that these models are not included in the **PyBoolNet** and **BBM** repositories. We then applied **bioLQM**, **PyBoolNet**, **mpbn**, and the

four variants of **Trappist** to computing minimal trap spaces of these real-world models. Table 2 shows the obtained experimental results. A number in bold indicates a ratio greater than or equal to 10 compared to the best result. The remaining notations are similar to those in Table 1. Hereafter, we analyze in detail the results with respect to minimal trap space computation.

Table 2: 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	CP	ILP	ASP
1 metastatic [44]	10	4	0.10	0.04	NM	0.01	1.15	0.89	0.02
2 Arabidopsis.thaliana [44]	15	8	0.10	0.06	NM	0.01	2.06	1.83	0.02
3 p53_high_dna [44]	16	1	0.38	1.76	NM	0.08	0.53	0.43	0.14
4 p53_low_dna [44]	16	1	0.41	1.76	NM	0.07	0.58	0.48	0.14
5 FT-GRN [45]	23	32	DNF	DNF	NM	0.03	8.41	12.38	0.19
6 DNA_damage [44]	26	16	0.24	0.33	NM	0.02	3.91	5.33	0.05
7 Rho-GTPases [44]	33	2	0.17	0.57	40.39	0.07	0.74	0.56	0.11
8 Pluripotency [46]	36	440	DNF	DNF	NM	0.16	138.92	DNF	0.28
9 Pluripotent [44]	36	276	0.37	0.43	NM	0.07	72.40	DNF	0.06
10 Pancreatic.Cancer [44]	43	1000+	N/A	0.11	0.36	0.17	DNF	DNF	0.06
11 Drosophila [47]	52	128	0.33	0.05	0.07	0.06	32.66	126.22	0.05
12 Cacace.TdevModel [48]	61	28	1.29	5.67	NM	0.06	7.51	23.15	0.08
13 hedgehog [44]	65	1000+	N/A	DNF	0.50	0.34	DNF	DNF	0.33
14 EMT [19]	69	268	39.22	1.01	0.20	0.12	75.81	DNF	0.05
15 Bcell [49]	73	72	0.23	0.04	0.08	0.06	18.95	81.85	0.05
16 mast_cell [6]	73	1000+	N/A	0.09	0.55	0.37	DNF	DNF	0.15
17 Corral.ThIL17diff [41]	92	1000+	N/A	107.57	0.76	0.56	DNF	DNF	0.16
18 Adhesion.CIP [50]	121	78	56.81	4.25	0.23	0.17	25.20	DNF	0.19
19 EMT_Mech [51]	136	82	DNF	14.01	0.27	0.20	27.55	DNF	0.25
20 macrophage [44]	136	1000+	N/A	0.54	1.09	0.84	DNF	DNF	0.27
21 angiogenesis [44]	141	1000+	N/A	0.16	1.07	1.06	DNF	DNF	0.16
22 angiofull [52]	142	1000+	N/A	0.17	1.06	0.88	DNF	DNF	0.23
23 EMT_Mech_TGFbeta [51]	150	492	DNF	11.28	0.78	0.69	DNF	DNF	0.35
24 RA_apoptosis [6]	180	1000+	N/A	DNF	1.43	1.55	DNF	DNF	0.19
25 MAPK [6]	181	1000+	N/A	13.58	1.76	1.51	DNF	DNF	0.27
26 Snf1-pathway [53]	202	1000+	N/A	1.13	1.47	1.43	DNF	DNF	0.31
27 T-cell-co-receptor [44]	206	1000+	N/A	DNF	1.52	2.26	DNF	DNF	0.35
28 TcellCheckPoint [54]	218	1000+	N/A	4.99	NM	1.96	DNF	DNF	0.28
29 Mycobacterium [44]	317	1000+	N/A	0.42	2.36	4.91	DNF	DNF	0.44
30 Leishmania [44]	342	1000+	N/A	DNF	2.56	5.62	DNF	DNF	0.46
31 Cholocystokinin [6]	383	1000+	N/A	0.36	2.99	4.81	DNF	DNF	0.37
32 Alzheimer [6]	762	1000+	N/A	DNF	NM	18.21	DNF	DNF	0.79

First, we obtained some observations on the four variants of **Trappist**

729 consistent with the observations obtained in the previous subsections. More
730 specifically, **Trappist-ASP** is still the best variant with a running time below
731 one second for every model, and followed by **Trappist-MaxSAT**. In particular,
732 the difference in running time between **Trappist-ASP** and **Trappist-MaxSAT**
733 is bigger for larger models or models with more than 1000 minimal trap
734 spaces. **Trappist-CP** and **Trappist-ILP** still have a much worse perfor-
735 mance, with **Trappist-CP** better than **Trappist-ILP**. They still can handle
736 no model with more than 1000 minimal trap spaces. However, **Trappist-CP**
737 or **Trappist-ILP** can handle the FT-GRN and Pluripotency models, whereas
738 all **bioLQM**, **PyBoolNet**, and **mpbn** cannot.

739 Second, **Trappist-ASP** (even **Trappist-MaxSAT**) is far more efficient than
740 both **bioLQM** and **PyBoolNet** on every model where the comparison is possi-
741 ble. For most models, the speedups of **Trappist-ASP** compared to **bioLQM**
742 and **PyBoolNet** are between one and three orders of magnitude. This again
743 confirms the superiority of **Trappist-ASP** compared to the other methods
744 that can handle general Boolean networks.

745 Third, for 11 of the 32 models (more than 34%), **mpbn** did not give any
746 answer because these models are non-locally-monotonic. For 21 of the 32 mod-
747 els where **mpbn** returned the answers, **mpbn** and **Trappist-ASP** are roughly
748 comparable in computation time, but **mpbn** appears quite slower on aver-
749 age. In particular, for the Rho-GTPases model, **mpbn** is $577\times$ slower than
750 **Trappist-ASP**. This observation along with the comparisons between **mpbn**
751 and **Trappist-ASP** in the previous subsections are quite surprising because
752 the ASP encoding of **mpbn** only requires the DNF for the activation part of a
753 Boolean function, whereas that of **Trappist-ASP** requires both the activation
754 and inhibition parts (see Subsection 4.3). However, the reason may lie on the
755 differences in the ASP encoding characteristics of the two methods and the
756 fact that **mpbn** needs to spend time checking the locally-monotonicity of each
757 Boolean function in a Boolean network. We expect that **mpbn** may outper-
758 form **Trappist** for a certain set of models, but not for the set of real-world
759 models considered in this article.

760 Fourth, regarding the comparison of the ASP-based methods (i.e., **PyBoolNet**,
761 **mpbn**, and **Trappist-ASP**), we note that for all the models where **PyBoolNet**
762 did not finish before the time limit, the timeout occurred during the compu-
763 tation of the prime-implicants. Hence, not even a single minimal trap space
764 was output by that method. For all the remaining models, once **PyBoolNet**
765 went through the prime-implicant phase, its ASP solving phase quickly re-
766 turned the first 1000 minimal trap spaces, all under one second. Hence,

767 with the experimental results shown in this subsection as well as the two
768 previous subsections, the practical differences between the ASP encoding of
769 **Trappist-ASP** and that of **PyBoolNet** are not distinctly exposed. The fact
770 that our new ASP encoding is guaranteed to be linear in the number of nodes
771 of the original model (see Subsection 4.3) does not seem to be crucial here,
772 however a much deeper analysis of those cases shall be shown in the next
773 subsection.

774 6.4. Randomly generated models

775 We randomly generated a set of N-K models [1] with network size n in the
776 set $\{100, 150, 200, 250, 300, 350, 400\}$ and in-degree $K = 3$ (i.e., each node
777 has exactly three input nodes). We chose N-K models because they are a
778 useful tool for studying the dynamics of Boolean networks [1, 7, 19]. For each
779 network size, 50 instances were generated using the `generateRandomNKNetwork`
780 function. In total, we have 350 random models. We then applied the com-
781 pared methods to these models and recorded the running time of each method
782 for each model. It is worth noting that N-K models usually have small
783 numbers of minimal trap spaces [7]. Hence, we searched for all solutions
784 in each model, which makes the comparison to **bioLQM** more comprehen-
785 sive. In addition, each node has only three input nodes, i.e., the number
786 of prime-implicants of the associated Boolean function is small. Hence,
787 **PyBoolNet** always passed the phase of computing prime-implicants in ev-
788 ery model even within one second, which enables us to compare the ASP
789 encoding of **PyBoolNet** and that of **Trappist-ASP**.

790 Figure 3 shows cumulative numbers of random models solved by the com-
791 pared methods with respect to enumerating all the minimal trap spaces. The
792 number of succeeded models within three minutes for each method is: **bioLQM**
793 (0), **PyBoolNet** (320), **mpbn** (0), **Trappist-maxSAT** (338), **Trappist-CP** (226),
794 **Trappist-ILP** (39), **Trappist-ASP** (349). We can see that **Trappist-ASP** is
795 the only method that can handle every model, but one. Note that none of
796 the other methods can handle that only model failed by **Trappist-ASP**. We
797 also obtained some observations consistent with those obtained for real-world
798 models. More specifically, **Trappist-MaxSAT** is still the second best method
799 and **Trappist-CP** is better than **Trappist-ILP**. Upon closer inspection, we
800 obtained several notable observations as follows.

801 First, **mpbn** was not able to handle any model because all the models
802 are non-locally-monotonic. Recall that a Boolean network is non-locally-
803 monotonic if only one of its Boolean functions is non-locally-monotonic.

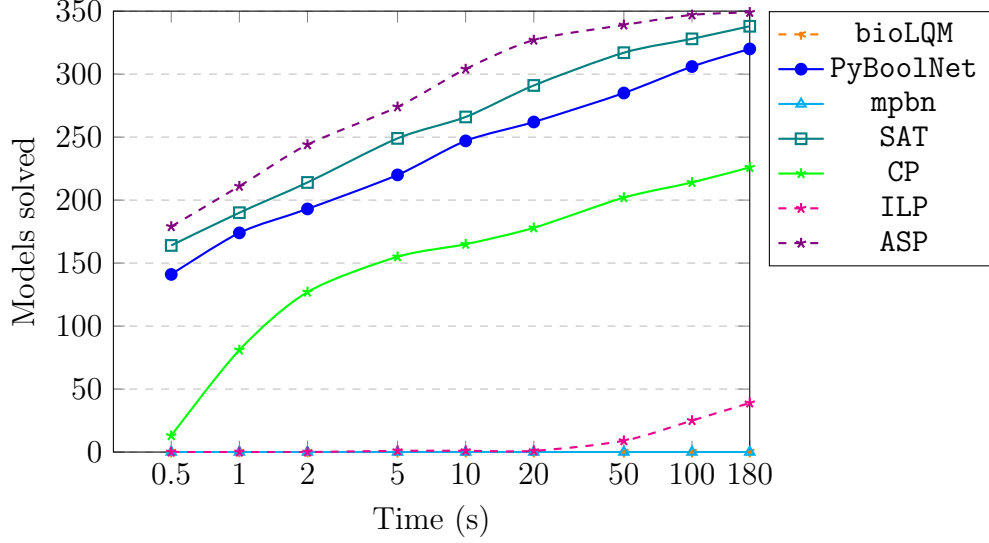


Figure 3: Cumulative numbers of random models solved by the compared methods with respect to enumerating all the minimal trap spaces.

Hence, it is apparent that all this type of randomly generated models are non-locally-monotonic because of the number of nodes is large ($n \geq 100$). This observation confirms a limit on the applicable model class of `mpbn`.

Second, surprisingly `bioLQM` cannot handle any model. One of the reason may be that the BDD characterizing all trap spaces is too large, and its computation is slow. In addition, having too many generic trap spaces before the filtering process may be also a reason. It is apparent because the network size is large ($n \geq 100$) and the Boolean functions are not simple.

Third, for every time limit, `Trappist-ASP` can always handle many more models than `PyBoolNet`, ranging from 29 to 65 more models. Since the time for the phase of computing prime-implicants of `PyBoolNet` is negligible in every model, most of the running time of `PyBoolNet` was spent for its ASP solving phase. Hence, we can easily see that the ASP encoding of `Trappist-ASP` is much better than that of `PyBoolNet`. This observation is consistent with the theoretical comparison in the ASP encoding between `Trappist-ASP` and `PyBoolNet` mentioned in Subsection 4.3.

6.5. Experimental summary

We have tested our alternative approach on many Boolean network models of various sizes and types (e.g., real-world models, randomly generated

models) on existing and newly created benchmarks. This indicates the high coverage and comprehensiveness of the experiments.

Among the four variants of the alternative approach, **Trappist-ASP** is the best method as it vastly outperforms all the other variants. The second best one is **Trappist-MaxSAT**. The two remaining variants (i.e., **Trappist-CP** and **Trappist-ILP**) give bad performance for most models. However, for certain cases, they are still better than all state-of-the-art methods (i.e., **bioLQM**, **PyBoolNet**, and **mpbn**). This is evidence for the advantages of an alternative approach compared to what preexisted.

Regarding general Boolean networks, **Trappist-ASP** (even **Trappist-MaxSAT**) is far more efficient than both **bioLQM** and **PyBoolNet**. The speedups of **Trappist-ASP** or **Trappist-MaxSAT** are large, even between one and three orders of magnitude for most models. In addition, the experimental results also confirm that the ASP encoding of **Trappist-ASP** is much more efficient than that of **PyBoolNet**.

Regarding locally-monotonic Boolean networks, the performance of **mpbn** is roughly comparable to that of **Trappist-ASP** or **Trappist-MaxSAT**. However, **mpbn** is quite slower than **Trappist-ASP** on average. This shows the practical advantage of **Trappist-ASP** compared to **mpbn**, though its ASP encoding may be more complex than that of **mpbn** in theory.

7. Conclusion

In this article we have explored and proved for the first time the equivalence between (minimal) trap spaces of a general Boolean network and (maximal) conflict-free siphons of its Petri net encoding. We have shown several useful applications of this finding to studying properties of trap spaces in Boolean networks. As an important practical application of the equivalence, we have proposed a new approach for the computation of minimal trap spaces in Boolean networks, based on the enumeration of maximal conflict-free siphons of Petri nets. We have also proposed the four possible methods using MaxSAT, CP, ILP, and ASP for implementing the new approach. In particular, we have shown how to adjust our approach to compute several specific types of trap spaces (e.g., maximal trap spaces, fixed points), which besides minimal trap spaces also play crucial roles in the analysis and control of Boolean networks. The proposed methods for the minimal trap space computation have been evaluated on many real-world models from the literature as well as randomly generated models. The experimental results show

859 that the new approach vastly outperforms all the state-of-the-art methods
 860 in terms of general Boolean networks and is comparable to the `mpbn` method
 861 even much better on average in terms of locally-monotonic Boolean net-
 862 works. We believe that this opens up the way to a much better analysis
 863 of large Boolean networks, which is needed with the advent of automatic
 864 model-generation pipelines [55].

865 Although the experimental results show the superiority of our approach
 866 to `mpbn` in general, we however note that there is a model in the `BBM` repos-
 867 itory (with identifier 122) where all the four proposed methods for the new
 868 approach did not manage to finish the Petri net conversion before the time-
 869 out, whereas `mpbn` can still handle this model. The model is not very large
 870 but its Boolean functions are rather complicated. This points to the fact that
 871 our current choice of using a BDD-based translation to obtain that Petri net
 872 encoding, though it provides a small/efficient ASP might be too costly to
 873 handle the complex models. In such a case, a more *naive* encoding might
 874 provide a much larger ASP program, with many redundant rules, but eas-
 875 ier/faster to obtain. The evaluation of the feasibility of such strategy, and
 876 of its impact on smaller instances, remains to be done. Recognizing that
 877 a model is locally-monotonic and applying in that specific case dedicated
 878 strategies as those of `mpbn` might also be a partial solution.

879 It is worth noting that there may be possibly other methods for comput-
 880 ing minimal/maximal conflict-free siphons in Petri nets, like the methods for
 881 generic siphon computation in the field of Petri nets (see [34] for a survey
 882 about these methods). Although these approaches do not directly support
 883 the minimal/maximal conflict-free siphon computation now, we plan to in-
 884 vestigate them in the future. They could replace our proposed methods if
 885 they give significantly better performance. However, the current methods
 886 appear to already perform very well even on the biggest models we have
 887 considered.

888 Finally, we think that the links between Petri nets and Boolean networks
 889 that we stumbled upon in this method might have deeper roots. Exploring
 890 those connections might lead both to interesting topics of research for Petri
 891 nets, like a notion of trap-spaces, and for Boolean networks. We also believe
 892 that the connection between trap spaces of Boolean networks and siphons
 893 of Petri nets can be a very useful tool for exploring and proving more new
 894 properties of trap spaces in Boolean networks, as we have used it to success-
 895 fully prove the independence of trap spaces to the update scheme and the
 896 separation of minimal trap spaces. Diving into this direction is promising

897 and one of our future work.

898 References

- 899 [1] L. Glass, S. A. Kauffman, The logical analysis of continuous, non-linear
900 biochemical control networks, *J. Theor. Biol.* 39 (1973) 103–129.
- 901 [2] R. Thomas, Boolean formalisation of genetic control circuits, *J. Theor.*
902 *Biol.* 42 (1973) 565–583.
- 903 [3] R. Thomas, R. d’Ari, Biological feedback, CRC press, 1990.
- 904 [4] R. Thomas, Regulatory networks seen as asynchronous automata: a
905 logical description, *J. Theor. Biol.* 153 (1991) 1–23.
- 906 [5] R.-S. Wang, A. Saadatpour, R. Albert, Boolean modeling in systems
907 biology: an overview of methodology and applications, *Phys. Biol.* 9
908 (2012) 055001.
- 909 [6] S. S. Aghamiri, V. Singh, A. Naldi, T. Helikar, S. Soliman, A. Niarakis,
910 J. Xu, Automated inference of Boolean models from molecular interac-
911 tion maps using CaSQ, *Bioinform.* 36 (2020) 4473–4482.
- 912 [7] H. Klarner, A. Bockmayr, H. Siebert, Computing maximal and minimal
913 trap spaces of Boolean networks, *Nat. Comput.* 14 (2015) 535–544.
- 914 [8] S. Chevalier, C. Froidevaux, L. Paulevé, A. Y. Zinovyev, Synthesis of
915 Boolean networks from biological dynamical constraints using answer-
916 set programming, in: *International Conference on Tools with Artificial*
917 *Intelligence*, IEEE, 2019, pp. 34–41.
- 918 [9] L. Paulevé, J. Kolčák, T. Chatain, S. Haar, Reconciling qualitative,
919 abstract, and scalable modeling of biological networks, *Nat. Commun.*
920 11 (2020) 1–7.
- 921 [10] M. Noual, D. Regnault, S. Sené, About non-monotony in Boolean au-
922 tomata networks, *Theor. Comput. Sci.* 504 (2013) 12–25.
- 923 [11] J. L. Peterson, Petri net theory and the modeling of systems, Prentice
924 Hall PTR, 1981.

- 925 [12] T. Murata, Petri nets: Properties, analysis and applications, Proc.
926 IEEE 77 (1989) 541–580.
- 927 [13] V. N. Reddy, M. L. Mavrovouniotis, M. N. Liebman, Petri net rep-
928 resentations in metabolic pathways, in: International Conference on
929 Intelligent Systems for Molecular Biology, AAAI, 1993, pp. 328–336.
- 930 [14] I. Zevedei-Oancea, S. Schuster, Topological analysis of metabolic net-
931 works based on Petri net theory, *Silico Biol.* 3 (2003) 323–345.
- 932 [15] M. A. Blätke, M. Heiner, W. Marwan, Biomodel engineering with Petri
933 nets, in: *Algebraic and Discrete Mathematical Methods for Modern*
934 *Biology*, Elsevier, 2015, pp. 141–192.
- 935 [16] O. Oanea, H. Wimmel, K. Wolf, New algorithms for deciding the siphon-
936 trap property, in: *International Conference on Applications and Theory*
937 *of Petri Nets*, Springer, 2010, pp. 267–286.
- 938 [17] F. Nabli, T. Martinez, F. Fages, S. Soliman, On enumerating mini-
939 mal siphons in Petri nets using CLP and SAT solvers: theoretical and
940 practical complexity, *Constraints An Int. J.* 21 (2016) 251–276.
- 941 [18] V. Trinh, B. Benhamou, K. Hiraishi, S. Soliman, Minimal trap spaces of
942 logical models are maximal siphons of their Petri net encoding, in: *In-*
943 *ternational Conference on Computational Methods in Systems Biology*,
944 Springer, 2022, pp. 158–176.
- 945 [19] J. C. Rozum, J. G. T. Zañudo, X. Gan, D. Deritei, R. Albert, Parity
946 and time reversal elucidate both decision-making in empirical models
947 and attractor scaling in critical Boolean networks, *Sci. Adv.* 7 (2021)
948 eabf8124.
- 949 [20] H. Klarner, A. Streck, H. Siebert, PyBoolNet: a python package for the
950 generation, analysis and visualization of Boolean networks, *Bioinform.*
951 33 (2017) 770–772.
- 952 [21] L. C. Fontanals, E. Tonello, H. Siebert, Control strategy identification
953 via trap spaces in Boolean networks, in: *International Conference on*
954 *Computational Methods in Systems Biology*, Springer, 2020, pp. 159–
955 175.

- 956 [22] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, S. Schwoon, Characteriza-
957 tion of reachable attractors using Petri net unfoldings, in: International
958 Conference on Computational Methods in Systems Biology, Springer,
959 2014, pp. 129–142.
- 960 [23] C. Chaouiya, E. Remy, P. Ruet, D. Thieffry, Qualitative modelling of
961 genetic networks: From logical regulatory graphs to standard Petri nets,
962 in: International Conference on Applications and Theory of Petri Nets,
963 Springer, 2004, pp. 137–156.
- 964 [24] C. Chaouiya, A. Naldi, E. Remy, D. Thieffry, Petri net representation of
965 multi-valued logical regulatory graphs, *Nat. Comput.* 10 (2011) 727–750.
- 966 [25] T. Chatain, S. Haar, J. Kolcák, L. Paulevé, A. Thakkar, Concurrency
967 in Boolean networks, *Nat. Comput.* 19 (2020) 91–109.
- 968 [26] C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, et al., SBML
969 qualitative models: a model representation format and infrastructure to
970 foster interactions between qualitative modelling formalisms and tools,
971 *BMC Syst. Biol.* 7 (2013) 1–15.
- 972 [27] S. M. Keating, D. Waltemath, M. König, F. Zhang, et al., SBML Level
973 3: an extensible format for the exchange and reuse of biological models,
974 *Mol. Syst. Biol.* 16 (2020) e9110.
- 975 [28] C. Chaouiya, A. Naldi, D. Thieffry, Logical modelling of gene regulatory
976 networks with GINsim, in: *Bacterial Molecular Networks*, Springer,
977 2012, pp. 463–479.
- 978 [29] A. Naldi, P. T. Monteiro, C. Müssel, C. for Logical Models, Tools,
979 H. A. Kestler, D. Thieffry, I. Xenarios, J. Saez-Rodriguez, T. Helikar,
980 C. Chaouiya, Cooperative development of logical modelling standards
981 and tools with CoLoMoTo, *Bioinform.* 31 (2015) 1154–1159.
- 982 [30] C. Müssel, M. Hopfensitz, H. A. Kestler, BoolNet - an R package for
983 generation, reconstruction and analysis of Boolean networks, *Bioinform.*
984 26 (2010) 1378–1380.
- 985 [31] D. Angeli, P. D. Leenheer, E. Sontag, A Petri net approach to persistence
986 analysis in chemical reaction networks, in: *Biology and Control Theory:
987 Current Challenges*, Springer, 2007, pp. 181–216.

- [32] D. Angeli, P. D. Leenheer, E. D. Sontag, Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws, *SIAM J. Appl. Math.* 71 (2011) 128–146.
- [33] E. Degrand, F. Fages, S. Soliman, Graphical conditions for rate independence in chemical reaction networks, in: *International Conference on Computational Methods in Systems Biology*, Springer, 2020, pp. 61–78.
- [34] G. Liu, K. Barkaoui, A survey of siphons in Petri nets, *Inf. Sci.* 363 (2016) 198–220.
- [35] V. Trinh, K. Hiraishi, B. Benhamou, Computing attractors of large-scale asynchronous Boolean networks using minimal trap spaces, in: *ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, ACM, 2022, pp. 13:1–13:10.
- [36] E. Demirović, G. Chu, P. J. Stuckey, Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers, in: *International Conference on Principles and Practice of Constraint Programming*, Springer, 2018, pp. 99–108.
- [37] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, MiniZinc: Towards a standard CP modelling language, in: *International Conference on Principles and Practice of Constraint Programming*, Springer, 2007, pp. 529–543.
- [38] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient MaxSAT solver, *J. Satisf. Boolean Model. Comput.* 11 (2019) 53–64.
- [39] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The Potsdam answer set solving collection, *AI Commun.* 24 (2011) 107–124.
- [40] J. Forrest, T. Ralphs, H. G. Santos, S. Vigerske, J. Forrest, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, rlougee, jpngoncall, Jan-Willem, h-i gassmann, S. Brito, Cristina, M. Saltzman, tosttost, B. Pitrus, F. MATSUSHIMA, to st, coin-or/Cbc: Release releases/2.10.8, 2022. URL: <https://doi.org/10.5281/zenodo.6522795>.

- 1019 [41] K. F. Corral-Jara, C. Chauvin, W. Abou-Jaoudé, M. Grandclaudon,
1020 A. Naldi, V. Soumelis, D. Thieffry, Interplay between SMAD2 and
1021 STAT5A is a critical determinant of IL-17A/IL-17F differential expres-
1022 sion, *Mol. Biomed.* 2 (2021) 1–16.
- 1023 [42] V. Singh, M. Ostaszewski, G. D. Kalliolias, G. Chiocchia, R. Olasso,
1024 E. Petit-Teixeira, T. Helikar, A. Niarakis, Computational systems bi-
1025 ology approach for the study of rheumatoid arthritis: from a molecular
1026 map to a dynamical model, *Genom. Comput. Biol.* 4 (2018) 100050.
- 1027 [43] S. Ogishima, S. Mizuno, M. Kikuchi, A. Miyashita, R. Kuwano,
1028 H. Tanaka, J. Nakaya, AlzPathway, an updated map of curated sig-
1029 naling pathways: towards deciphering Alzheimer’s disease pathogenesis,
1030 in: *Systems Biology of Alzheimer’s Disease*, Springer, 2016, pp. 423–432.
- 1031 [44] C. Kadelka, T.-M. Butrie, E. Hilton, J. Kinseth, H. Serdarevic, A meta-
1032 analysis of Boolean network models reveals design principles of gene
1033 regulatory networks, *arXiv preprint arXiv:2009.01216* (2020).
- 1034 [45] E. C. Chávez-Hernández, S. Quiroz, B. García-Ponce, E. R. Álvarez-
1035 Buylla, The flowering transition pathways converge into a complex gene
1036 regulatory network that underlies the phase changes of the shoot apical
1037 meristem in *Arabidopsis thaliana*, *Front. Plant Sci.* 13 (2022) 852047.
- 1038 [46] A. Yachie-Kinoshita, K. Onishi, J. Ostblom, M. A. Langley, E. Posfai,
1039 J. Rossant, P. W. Zandstra, Modeling signaling-dependent pluripotency
1040 with Boolean logic to predict cell fate transitions, *Mol. Syst. Biol.* 14
1041 (2018) e7952.
- 1042 [47] M. R. Vega, Analyzing toys models of *Arabidopsis* and *Drosophila* us-
1043 ing Z3 SMT-LIB, in: *Independent Component Analyses, Compressive
1044 Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII*,
1045 volume 9118, SPIE, 2014, pp. 240–254.
- 1046 [48] E. Cacace, S. Collombet, D. Thieffry, Logical modeling of cell fate
1047 specification—Application to T cell commitment, in: *Current Topics in
1048 Developmental Biology*, Elsevier, 2020, pp. 205–238.
- 1049 [49] P. Dutta, L. Ma, Y. Ali, P. M. Slood, J. Zheng, Boolean network model-
1050 ing of B-cell apoptosis and insulin resistance in type 2 diabetes mellitus,
1051 *BMC Syst. Biol.* 13 (2019) 1–12.

- 1052 [50] E. Guberman, H. Sherief, E. R. Regan, Boolean model of anchorage
1053 dependence and contact inhibition points to coordinated inhibition but
1054 semi-independent induction of proliferation and migration, *Comput.*
1055 *Struct. Biotechnol. J.* 18 (2020) 2145–2165.
- 1056 [51] E. Sullivan, M. Harris, A. Bhatnagar, E. Guberman, I. Zonfa, E. R. Re-
1057 gan, Boolean modeling of mechanosensitive Epithelial to Mesenchymal
1058 Transition and its reversal, *bioRxiv* (2022).
- 1059 [52] N. Weinstein, L. Mendoza, I. Gitler, J. Klapp, A network model to
1060 explore the effect of the micro-environment on endothelial cell behavior
1061 during angiogenesis, *Front. Physiol.* 8 (2017) 960.
- 1062 [53] T. Lubitz, N. Welkenhuysen, S. Shashkova, L. Bendrioua, S. Hohmann,
1063 E. Klipp, M. Krantz, Network reconstruction and validation of the
1064 Snf1/AMPK pathway in baker’s yeast based on a comprehensive litera-
1065 ture review, *npj Syst. Biol. Appl.* 1 (2015) 1–10.
- 1066 [54] C. Hernandez, M. Thomas-Chollier, A. Naldi, D. Thieffry, Computa-
1067 tional verification of large logical models—Application to the prediction
1068 of T cell response to checkpoint inhibitors, *Front. Physiol.* 11 (2020)
1069 558606.
- 1070 [55] M. Ostaszewski, A. Niarakis, A. Mazein, I. Kuperstein, R. Phair,
1071 A. Orta-Resendiz, V. Singh, S. S. Aghamiri, M. L. Acencio, E. Glaab,
1072 et al., COVID19 Disease Map, a computational knowledge repository of
1073 virus–host interaction mechanisms, *Mol. Syst. Biol.* 17 (2021) e10387.