Minimal trap spaces of Logical models are maximal siphons of their Petri net encoding

Van-Giang Trinh[1][0000−0001−6581−998X],
Belaid Benhamou[1],
Kunihiko Hiraishi[2][0000−0003−1750−1891], and
Sylvain Soliman[3][0000−0001−5525−7418]

[1] LIS, Aix-Marseille Université, Marseille, France
{trinh.van-giang, belaid.benhamou}@lis-lab.fr
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Japan
hira@jaist.ac.jp
[3] Lifeware team, Inria Saclay center, Palaiseau, France
Sylvain.Soliman@inria.fr

Abstract. Boolean modelling 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. 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. 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, the state-of-the-art computation of minimal trap spaces based on *prime-implicants* shows its limits as there can be a huge number of implicants.

In this article we present an alternative method to compute minimal trap spaces, and hence complex attractors, of a Boolean model. 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. After some technical preliminaries, we expose the concrete need for such a method and detail its implementation using Answer Set Programming. We then demonstrate its efficiency and compare it to implicant-based methods on some large Boolean models from the literature.

Keywords: Logical models · Boolean models · Trap spaces · Attractor computation · Petri nets · Siphons

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 [18,46,48,47].

Boolean modelling 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 [50], 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 [1]. 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 [27] 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, the state-of-the-art computation of minimal trap spaces based on *prime-implicants* shows its limits as there can be a huge number of implicants.

It is worth noting that the recent method presented in [11] for computing minimal trap spaces avoids the prime-implicants computation by relying on the *most-permissive* semantics of Boolean models. This method has been implemented in the tool mpbn[4] demonstrated in [41] 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 models, whereas the prime-implicants based method [27] is applicable for *general* Boolean models (i.e., including both locally-monotonic and non-locally-monotonic Boolean models). The study [37] highlights the need for non-locally-monotonic Boolean models in both biological and theoretical aspects. Hence, it is still necessary to develop efficient methods for computing minimal trap spaces of large-scale general Boolean models.

Petri nets were introduced in the 60s as simple formalism for describing and analyzing information-processing systems that are characterized as being concurrent, asynchronous, non-deterministic and possibly distributed [42,33]. The use of Petri nets for representing biochemical reaction systems, by mapping molecular species to places and reactions to transitions, hinted at already in [42,33] was used more thoroughly quite late in [43], together with some Petri net concepts and tools for the analysis of metabolic networks. Siphons are such a concept, but they have not been used a lot for the study of biochemical systems [51,4] even if the practical cost of computing their minimal/maximal elements appear much more manageable than the theoretical complexity would indicate [38,35].

In this article we present an alternative method to compute minimal trap spaces, and hence complex attractors, of a general Boolean model. 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. Af-

[4] https://github.com/bnediction/mpbn

ter some technical preliminaries, we expose the concrete need for such a method and detail its implementation using Answer Set Programming. We then demonstrate its efficiency and compare it to implicant-based methods on several large Boolean models from the literature.

All models used for evaluation and the implementation of the presented method are available at https://github.com/soli/trap-spaces-as-siphons and executable as a CoLoMoTo docker image.

2 Preliminaries

We will briefly recall here some preliminaries on Boolean models related to trap spaces and Petri nets. In the case of multi-level Logical models, an encoding into a Boolean model is always possible [15].

2.1 Traps spaces

We recall here some definitions from [27] for the introduction of *trap spaces*. Minimal trap spaces prove to be a very good approximation of the attractors of a Boolean model under asynchronous update schemes and have become the *de facto* standard way to analyze models of a few tens of *genes* [28,16].

Given a Boolean model $\mathcal{M} = (V, F)$ with nodes $V = (v_1, \ldots, v_n)$ and Boolean functions $F = (f_1, \ldots, f_n)$, its state-space is $\mathcal{S}_\mathcal{M} = \mathbb{B}^n$ with $\mathbb{B} = \{0, 1\}$. A state $s \in \mathbb{B}^n$ is a mapping $s : V \mapsto \mathbb{B}$ that assigns either 0 (inactive) or 1 (active) to each node. At each time step t, node v_i can update its state by

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

where $v(t)$ is the state of $\mathcal{M}$ at time t and $v_i(t+1)$ the state of node v_i at time $t+1$. An update scheme of a Boolean model specifies the way that the nodes of the model update their states through time evolution [47]. There are two main types of update schemes: synchronous and asynchronous.

A non-empty set $T \subseteq S_\mathcal{M}$ is a *trap set* of $\mathcal{M}$ if and only if for every $x \in T$ and $y \in S_\mathcal{M}$ with y is reachable from x it holds that $y \in T$.

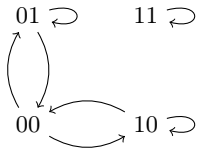
A *subspace* m of $\mathcal{S}_\mathcal{M}$ is characterized by its fixed nodes (denoted by D_m) and free nodes. This subspace can be specified by an assignment $m : D_m \mapsto \mathbb{B}$ where $D_m \subseteq V$ and $m(u)$ is the value of node $u \in D_m$. The remaining nodes of $\mathcal{M}$, $V \setminus D_m$, are said to be *free*, i.e., they can receive any Boolean value. We write subspaces like states but use in addition the symbol $\star$ to indicate that a node is free. A subspace m thus corresponds to the set of states $\mathcal{S}_\mathcal{M}[m] := \{s \in \mathcal{S}_\mathcal{M} \mid \forall v \in D_m : s(v) = m(v)\}$. For example, $m = \star\star1$ means that $D_m = \{v_3\}$, $m(v_3) = 1$, and corresponds to the set of states $\{001, 011, 101, 111\}$. Let $\mathcal{S}^\star_\mathcal{M}$ denote the set of all possible subspaces of $\mathcal{M}$. Note that $|\mathcal{S}^\star_\mathcal{M}| = 3^n$ and $S_\mathcal{M} \subset \mathcal{S}^\star_\mathcal{M}$ [27].

A *trap space* is defined as a subspace that is also a trap set. It is noted that trap spaces of a Boolean model are independent of the update scheme of this

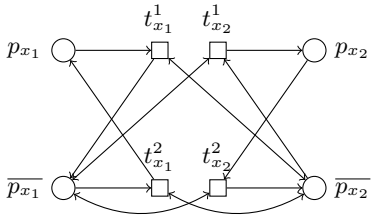
model [27]. Then, we define a partial order $<$ on $\mathcal{S}^\star_\mathcal{M}$ as: $m < m'$ if and only if $\mathcal{S}_\mathcal{M}[m] \subseteq \mathcal{S}_\mathcal{M}[m']$ and $\mathcal{S}_\mathcal{M}[m] \neq \mathcal{S}_\mathcal{M}[m']$. Consequently, a trap space m is minimal if and only if there is no trap space $m' \in \mathcal{S}^\star_\mathcal{M}$ such that $m' < m$.

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

Example 1. We give a Boolean model $\mathcal{M} = (V, F)$, where $V = (x_1, x_2)$ and $F = (f_1, f_2)$ with $f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2), f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$. Herein, $\wedge$, $\vee$, and $\neg$ denote the conjunction, disjunction, and negation logical operators, respectively.



(a) State transition graph, under the fully asynchronous update.



(b) Petri net encoding of the model. Circles denote places, whereas rectangles denote transitions.

Fig. 1: Dynamics and encoding of the Boolean model of Example 1

2.2 Petri net encoding of Boolean models

Definition 1. *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 $m(p) > 0$. We shall write $pred(x)$ (resp. $succ(x)$) to represent the set of vertices that have a (non-zero weighted) arc leading to (resp. coming from) x.

The link between Boolean models *à la* Thomas and Petri nets was originally established in [8] 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 models but was later extended to multivalued Logical models in two ways, either in [6] with non 1-safe Petri nets or more recently in [9] with 1-safe nets but many more places.

Since our study is focused on Boolean models, we briefly recall the original encoding here. Its basis is that every node (*gene*) v of the original model $\mathcal{M} = (V, F)$ is represented by two separate places (p_v and $\overline{p}_v$), corresponding to its two states, active, and inactive, respectively. Each conjunct of the logical function that activates the *gene* will lead to a transition t, consuming the inactive place (i.e., a directional arc from $\overline{p}_v$ to t), producing the active place (i.e., a directional arc from t to p_v), and with all other literals both consumed and produced (i.e., a bidirectional arc). And conversely for the inactivation. Let s be a state of the Boolean model and m_s be its corresponding marking in the encoded Petri net. It holds that $\forall v \in V$, $s(v) = 0$ if and only if $m_s(\overline{p}_v) = 1$ and $s(v) = 1$ if and only if $m_s(p_v) = 1$. Note also that at any marking m of the Petri net encoding a Boolean model, it always holds that $m(p_v) + m(\overline{p}_v) = 1$.

The main property of this encoding is that it is completely faithful with respect to the update scheme of the original Boolean model. For each node v of $\mathcal{M}$, only transitions corresponding to v can change the current marking of p_v or $\overline{p_v}$. In addition, at any marking at most one of such transitions is enabled because $m(p_v) + m(\overline{p}_v) = 1$ holds. Hence, for any update scheme in $\mathcal{M}$, we have a corresponding firing scheme in $\mathcal{P}$, which preserves the equivalence between the dynamics of $\mathcal{M}$ and $\mathcal{P}$ [10].

For illustration, let us reconsider the Boolean model shown in Example 1. Figure 1(b) shows the Petri net encoding of this Boolean model. Place p_{x_1} (resp. $\overline{p_{x_1}}$) in $\mathcal{P}$ represents the activation (resp. the inactivation) of node x_1 in $\mathcal{M}$. Marking $\{p_{x_1}, \overline{p_{x_2}}\}$ in $\mathcal{P}$ represents state 10 in $\mathcal{M}$. Transitions $t^1_{x_1}$ and $t^2_{x_1}$ represent the update of node x_1. Of course, in any marking $t^1_{x_1}$ and $t^2_{x_1}$ cannot be both enabled. Then, the fully asynchronous update scheme in $\mathcal{M}$ corresponds to the classical firing scheme in $\mathcal{P}$ where only one of the enabled transitions for a given marking will be fired [33].

Note that given a Boolean model in the standard SBML-Qual format [5], i.e., the package of SBML v3 [24] for such models, one can easily obtain its Petri net encoding in the Petri Net Markup Language (PNML)[5] standard using the BioLQM[6] library. This piece of software extracted from GINsim [7] and part of the CoLoMoTo[7] [36] software suite allows for easy conversion between standard formats. It also accepts many other common formats for Boolean models, notably the `.bnet` files of the BoolNet [34,28] tools. The conversion is executed as follows:

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

Note that transforming a Boolean model defined by its functions into its Petri net encoding roughly relies on obtaining conditions for the activation and inactivation of the states. In [8] this took the form of the whole truth table of the Boolean functions, but as shown in Appendix 1 of [9] 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. In practice

[5] https://www.pnml.org/
[6] http://www.colomoto.org/biolqm/
[7] http://colomoto.org/

this transformation, here using BDDs in BioLQM, seems to behave quite well on even quite big models as will be shown in the Section 6 on evaluation.

2.3 Siphons

Siphons are a static and classical property of Petri nets [42]. 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 [2,3,14].

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

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

Note that $\emptyset$ is trivially a siphon.

3 Minimal trap spaces as maximal conflict-free siphons

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

Definition 3. *A set of places of Petri net* $\mathcal{P}$ *encoding Boolean model* $\mathcal{M}$ *is* conflict-free *if it does not contain any two places corresponding to the active and inactive states of the same* gene *of* $\mathcal{M}$*. Then, a conflict-free siphon* S *is said to be* maximal *if and only if there is no other conflict-free siphon* S' *such that* $S \subset S'$*.*

Intuitively, a siphon is a set of places that once unmarked remains so. If it is conflict-free then its dual corresponds to a partial-state of the model such that whatever update, the fixed values remain so (since the unmarked places remain unmarked). This is precisely the definition of a trap space and maximality of the siphon is equivalent to as many fixed values as possible, hence minimality of the trap space. For example, the Boolean model given in Example 1 has two trap spaces, $m_1 = 11$ and $m_2 = \star\star$. The Petri net encoding of this Boolean model has five generic siphons, $S_1 = \emptyset$, $S_2 = \{p_{x_1}, \overline{p_{x_1}}\}$, $S_3 = \{p_{x_2}, \overline{p_{x_2}}\}$, $S_4 = \{\overline{p_{x_1}}, \overline{p_{x_2}}\}$, and $S_5 = \{p_{x_1}, \overline{p_{x_1}}, p_{x_2}, \overline{p_{x_2}}\}$. However, only S_1 and S_4 are conflict-free siphons and correspond to m_2 and m_1, respectively. Since $S_1 \subset S_4$, S_4 is a maximal siphon corresponding to the minimal trap space m_1. Hereafter, we formally prove that a maximal conflict-free siphon is equivalent to a minimal trap space.

Definition 4. *Let* m *be a subspace of Boolean model* $\mathcal{M} = (V, F)$*. A* mirror *of* m *is a set of places* S *in the Petri net encoding* $\mathcal{P}$ *of* $\mathcal{M}$ *such that:*

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

and

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

Theorem 1. *Let $\mathcal{M} = (V, F)$ be a Boolean model and $\mathcal{P}$ be its Petri net encoding. A subspace m is a trap space of $\mathcal{M}$ if and only if its mirror S is a conflict-free siphon of $\mathcal{P}$.*

Proof. First, we show that if m is a trap space of $\mathcal{M}$, then S is a conflict-free siphon of $\mathcal{P}$ (*). If $D_m = \emptyset$, then $S = \emptyset$ is trivially a conflict-free siphon of $\mathcal{P}$. Thus, we consider the case that $D_m \neq \emptyset$ (resp. $S \neq \emptyset$). Assume that S is not a siphon of $\mathcal{P}$. Then, there is a transition $t \in T$ such that $S \cap succ(t) \neq \emptyset$ but $S \cap pred(t) = \emptyset$. In other words, there is a place $p \in S$ such that $p \in succ(t)$ but $p \notin pred(t)$. Let v be the corresponding node in $\mathcal{M}$ of p. By the characterization of the encoding [8], there is a directional arc from t to p and a directional arc from the complementary place of p to t. Without loss of generality, we assume that $p = p_v$, then there is a directional arc from t to p_v and a directional arc from $\overline{p}_v$ to t. In addition, there is also no arc or a bidirectional arc between t and another place rather than p_v and $\overline{p}_v$. Thus, there is no connecting arc between t and any place in $S \setminus \{p_v\}$ because $S \cap pred(t) \neq \emptyset$. In $\mathcal{S}_\mathcal{M}[m]$, a node in $V \setminus D_m$ can receive any Boolean value. Hence, there is a state $s \in \mathcal{S}_\mathcal{M}[m]$ such that $m_s(p') = 1, \forall p' \in pred(t) \setminus \{\overline{p_v}\}$ where m_s is the corresponding marking in $\mathcal{P}$ of s. We also have $m_s(p_v) = 0$, leading to $m_s(\overline{p_v}) = 1$ by the characterization of the encoding [8]. Now, t is enabled at marking m_s. Its firing leads to a new marking m'_s such that $m'_s(p_v) = 1$ and $m'_s(\overline{p}_v) = 0$. Let s' be the corresponding state in $\mathcal{M}$ of m'_s. Since m is a trap space of $\mathcal{M}$, $s' \in \mathcal{S}_\mathcal{M}[m]$. Then, $s'(v) = m(v)$, leading to $m'_s(p_v) = 0$, which is a contradiction. Hence, S is a siphon of $\mathcal{P}$. By the definition of a mirror, S is also a conflict-free one.

Second, we show that if S is a conflict-free siphon of $\mathcal{P}$, then m is a trap space of $\mathcal{M}$ (**). By the definition of a mirror, m is a subspace of $\mathcal{M}$. Let s be an arbitrary state in $\mathcal{S}_\mathcal{M}[m]$ and m_s be its corresponding marking in $\mathcal{P}$. By the characterization of the encoding [8], $m_s(p) = 0, \forall p \in S$. In any marking m'_s reachable from m_s regardless of the firing scheme of $\mathcal{P}$, we have $m'_s(p) = 0, \forall p \in S$ by the dynamical property on markings of a siphon [31]. Equivalently, in any state s' reachable from s regardless of the update scheme of $\mathcal{M}$, we have $s'(v) = s(v) = m(v), \forall v \in D_m$. Then, $s' \in \mathcal{S}_\mathcal{M}[m]$. By the definition of a trap space and the arbitrariness of s, m is a trap space of $\mathcal{M}$.

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

Theorem 2. *Let $\mathcal{M}$ be a Boolean model and $\mathcal{P}$ be its Petri net encoding. A subspace m is a minimal trap space of $\mathcal{M}$ if and only if its mirror S is a maximal conflict-free siphon of $\mathcal{P}$.*

Proof. First, we show that if m is a minimal trap space of $\mathcal{M}$, then S is a maximal conflict-free siphon of $\mathcal{P}$ (*). Since m is a trap space of $\mathcal{M}$, S is a conflict-free siphon of $\mathcal{P}$ by Theorem 1. Assume that S is not maximal. Then, there is another conflict-free siphon S' such that $S \subset S'$. By Theorem 1, there is a trap space m' corresponding to S'. Following the definition of a mirror, $\mathcal{S}_\mathcal{M}[m'] \subset \mathcal{S}_\mathcal{M}[m]$, thus $m' < m$. This is a contradiction because m is a minimal trap space. Hence, S is a maximal conflict-free siphon of $\mathcal{P}$.

Second, we show that if S is a maximal conflict-free siphon of $\mathcal{P}$, then m is a minimal trap space of $\mathcal{M}$ (**). Since S is a conflict-free siphon of $\mathcal{P}$, m is a trap space of $\mathcal{M}$ by Theorem 1. Assume that m is not minimal. Then, there is another trap space m' such that $m' < m$. In other words, $\mathcal{S}_{\mathcal{M}}[m'] \subset \mathcal{S}_{\mathcal{M}}[m]$. Let S' be the mirror of m'. S' is a conflict-free siphon by Theorem 1. Following the definition of a mirror, $S \subset S'$, which is a contradiction because S is a maximal conflict-free siphon. Hence, m is a minimal trap space of $\mathcal{M}$.

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

By Theorem 2, we can reduce the problem of computing all minimal trap spaces of a Boolean model to the problem of computing all maximal conflict-free siphons of its Petri net encoding. Note that in the case of stable states, this can be put in regard to the classical relationship between siphons and deadlocks in Petri nets. It might actually be possible to generalize our result to any 1-safe place-complementary Petri net to define a notion of trap space that might be useful for the analysis of Petri nets, but this is out of the scope of this article.

It is noted that there are no existing methods specifically designed for computing maximal conflict-free siphons (even maximal siphons) of a Petri net. The reason might be that researchers mainly focus on minimal siphons [31]. Hence, we here propose a new method based on Answer Set Programming (ASP) [17] for computing maximal conflict-free siphons of a Petri net. The details of the proposed method shall be given in the next section.

4 Answer set programming-based method

First, we show the characterization of all conflict-free siphons of the encoded 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 definition all the transitions in $pred(p)$ must belong to $succ(S)$. Note that $succ(S) = \bigcup_{p \in S} succ(p)$. A transition t belongs to $succ(S)$ if and only if there is at least one place p' in S such that $p' \in pred(t)$. Hence, for each transition $t \in pred(p)$, we can state that

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

The system of all the rules of the above form with respect to all pairs (p, t) where $p \in P, t \in T, t \in pred(p)$ fully characterizes all generic siphons of a Petri net and has been used with SAT solvers in [38,35]. To make S to be a conflict-free siphon, we need to add to the system the rule

$$p_v \in S \Rightarrow \overline{p_v} \notin S \wedge \overline{p_v} \in S \Rightarrow p_v \notin S \tag{2}$$

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

Then, we translate the above characterization into the ASP $\mathcal{L}$ as follows. We introduce atom `p-v` (resp. `n-v`) to denote place p_v (resp. $\overline{p_v}$), $\forall v \in V$. The set

of all atoms in $\mathcal{L}$ is given as $\mathcal{A} = \bigcup_{v \in V} \{\texttt{p-v}, \texttt{n-v}\}$. For each pair (p, t) where $p \in P, t \in T, t \in pred(p)$, we translate the rule (1) into the ASP rule

```
a_1; ... ; a_k :- a.
```

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

```
:- p-v, n-v.
```

for each $v \in V$. This ASP rule guarantees that two places representing the same node in $\mathcal{M}$ never belong to the same siphon of $\mathcal{P}$, representing the conflict-freeness. Naturally, a Herbrand model (see, e.g, [17]) 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

```
{p-v}. {n-v}.
```

for each $v \in V$. Note that the number of atoms of $\mathcal{L}$ is only $2n$, whereas the ASP encoding shown in [27] has as many atoms as the number of prime-implicants of the Boolean model and that number might be exponential in n. In [11], there is an ASP characterization of trap spaces that does not rely on minimal DNFs either and thus seems very similar to our ASP encoding. Remarkably it only requires the DNF for the *activation* part, using the information that it will only be used for locally-monotonic Boolean models. We would therefore expect that, when available, it will have comparable performance on the ASP part (the ASP program would be twice smaller, but redundancy is not always bad in that field), but can also avoid combinatorial explosion of the Petri net encoding for some formula where the activation DNF is simple but the inhibition is not. Since mpbn is included in our benchmark this will be evaluated in our experiments.

Now, a solution (simply an answer set) $A \subseteq \mathcal{A}$ of $\mathcal{L}$ is equivalent to a conflict-free siphon S of $\mathcal{P}$, thus a trap space m of $\mathcal{M}$. The conversion from A to m is straightforward. If $\texttt{p-v} \in A$ then $v \in D_m$ and $m(v) = 0$. Conversely, if $\texttt{n-v} \in A$ then $v \in D_m$ and $m(v) = 1$. Otherwise, $v \notin D_m$. Computing multiple answer sets is built into ASP solvers and the solving collection POTASSCO [17] also features the option to find set-inclusion maximal answer sets with respect to the set of atoms. Naturally, a set-inclusion maximal answer set of $\mathcal{L}$ is equivalent to a maximal conflict-free siphon of $\mathcal{P}$, thus a minimal trap space of $\mathcal{M}$. By using this built-in option, we can compute all the set-inclusion maximal answer sets of $\mathcal{L}$ (resp. all the minimal trap spaces of $\mathcal{M}$) in one execution.

5 Motivating example

For a few years now we have been collaborating with biologists who build very large detailed and annotated maps and now wish to analyze the dynamics of the corresponding models. One of the main maps studied this way represents

knowledge about the Rheumatoïd Arthritis [45], and was the main motivation for the development of a tool to automatically transform it into an executable Boolean model [1]. In the supplementary material of the paper, an excerpt of the map, focused around the apoptosis (cell death) module is transformed into a model of *reasonable* size, namely 180 Boolean variables (model `F5_RA_apoptosis_executable_module.sbml` of supplementary material S3, and model "RA-apoptosis" of Section 6). The study of such model, though, is a big hurdle. Indeed, as stated in the article about another model of the same size: *"The size of the CaSQ-inferred MAPK model (181 nodes) made the calculation of stable states a non-realistic endeavour."*

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

However, using a state-of-the-art tool like PyBoolNet [27] on that model actually fails at the phase of prime-implicant generation. mpbn [41] does not give any answer either because it recognizes that model as non-locally-monotonic. And hence, it is not possible to extract any (complex) attractor at all. This is also true for the Alzheimer model also mentioned in that same article and originally from [39] (`F4` file in the original supplementary material, and "Alzheimer" in Table 2), but actually not for the MAPK model for which the first trap spaces can be obtained in reasonable time. The current practice usually revolves then around fixing some inputs to plausible values and reducing the model accordingly. While this approach makes sense, it relies on potentially arbitrary decisions, and *hides away* critical modelling choices that were actually not part of the original Boolean model or even of the starting map.

Using the method presented above, it is possible to convert the model to PNML in about one second and to obtain the first 1000 minimal trap spaces (including ones that contain more than one state) in a few milliseconds. Unfortunately since this was not available at the time, the analysis of the model remained very high-level and qualitative, instead of being able to use the rich information of computed minimal trap spaces.

6 Evaluation

To assess the efficiency of the proposed method, implemented as a Python package named Trappist, we compare it with the state-of-the-art method implemented in the tool PyBoolNet [27,28] on both its own repository of models and large models from the literature. In addition, we also include the tool mpbn [41] to the benchmarks although it only handle locally-monotonic models, whereas both Trappist and PyBoolNet can handle general models.

To solve the ASP problems, we used the same ASP solver CLINGO [17] and the same configuration as that used in PyBoolNet [27,28]. 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 an apple laptop whose environment is CPU: Intel® Core™ i7 1.20GHz x 4, 16 GB DDR4 RAM, MacOS 12.3.1. Finally, we set a time limit of two minutes for each model.

For all the above tools, namely PyBoolNet, mpbn and CLINGO, we used the version available in the latest CoLoMoTo docker image tagged `2022-05-01`. All the models and a CoLoMoTo notebook realizing the benchmarks can be found at https://github.com/soli/trap-spaces-as-siphons. These can be run on a Docker image in the cloud by clicking the "Binder" button.

6.1 PyBoolNet repository

As shown in Table 1, for most of the models of the official PyBoolNet repository[8], the results are comparable with all minimal trap spaces found very fast. For 5 of the 29 models, mpbn did not give any answer because it recognized these models as not locally-monotonic. Note that on some very small models, Trappist is sometimes slower than PyBoolNet and/or mpbn, but still significantly under one second. Moreover, we believe that the result on the arellano_rootstem model is caused by the cold start of the JVM for BioLQM. On the contrary, on every model that was a bit challenging for PyBoolNet or mpbn, the new method is far more efficient with speedups between one and two orders of magnitude.

6.2 Selected models

We used a set of real-world Boolean models lying in various scales collected from numerous bibliographic sources. 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 most of them have never been fully analyzed. We then applied PyBoolNet, mpbn, and Trappist to computing minimal trap spaces of these real-world models. It is notable that unlike existing analysis shown in the literature, we did not fix specific values for source nodes (i.e., some node v such that $f_v = v$) in these models. Table 2 shows the experimental results on those models. Hereafter, we analyze in detail the results with respect to minimal trap space computation.

The first observation is that for 26 of the 33 models (more than 78%), mpbn did not give any answer because it recognized that these models as not locally-monotonic. For 6 of the 33 models where mpbn returned the answers, mpbn and Trappist are comparable in computation time, though surprisingly a bit slower on average. Note however that mpbn was the only tool to provide a solution for the SN-5 model, thus confirming that if the activation function is in the right form, not having to compute the inactivation function's disjunctive normal form

[8] https://github.com/hklarner/pyboolnet/tree/master/pyboolnet/repository

Table 1: Timing comparisons between PyBoolNet, mpbn and Trappist on the PyBoolNet repository. Column n denotes the number of nodes of each model. Column $|M|$ denotes the number of minimal trap spaces and for each method is given the computation time in seconds, asking only for the first 1000 trap spaces. A number in bold indicates a ratio greater than three compared to the best result. "NM" indicates a non-locally-monotonic model.

	model	n	$\|M\|$	PyBoolNet	mpbn	Trappist
1	arellano_rootstem	9	4	0.05	0.01	**0.25**
2	calzone_cellfate	28	27	0.03	**NM**	0.03
3	dahlhaus_neuroplastoma	23	32	0.09	0.02	0.03
4	davidich_yeast	10	12	0.03	0.01	0.02
5	dinwoodie_life	15	7	0.03	0.01	0.01
6	dinwoodie_stomatal	13	1	0.03	0.01	0.01
7	faure_cellcycle	10	2	0.04	0.01	0.01
8	grieco_mapk	53	18	0.04	0.04	0.03
9	irons_yeast	18	1	0.04	0.01	0.02
10	jaoude_thdiff	103	>1000	**1.43**	**1.20**	0.06
11	klamt_tcr	40	8	0.04	0.02	0.01
12	krumsiek_myeloid	11	6	0.03	0.01	0.01
13	multivalued	13	4	0.02	0.01	0.01
14	n12c5	11	5	**35.21**	0.02	0.02
15	n3s1c1a	2	2	0.03	0.01	0.01
16	n3s1c1b	2	2	0.03	0.01	0.01
17	n5s3	4	3	0.03	**NM**	0.01
18	n6s1c2	5	3	0.04	0.01	0.01
19	n7s3	6	3	0.03	0.01	0.01
20	raf	3	2	0.03	0.01	0.01
21	randomnet_n15k3	15	3	0.04	**NM**	0.01
22	randomnet_n7k3	7	10	0.03	**NM**	0.01
23	remy_tumorigenesis	34	25	**2.41**	0.03	0.02
24	saadatpour_guardcell	13	1	0.05	0.01	0.01
25	selvaggio_emt	56	>1000	**1.75**	**0.64**	0.04
26	tournier_apoptosis	12	3	0.06	0.01	0.01
27	xiao_wnt5a	7	4	0.04	0.01	0.01
28	zhang_tlgl	60	156	0.33	**NM**	0.04
29	zhang_tlgl_v2	60	258	0.12	0.18	0.03

can render a difficult problem tractable. However, since mbpn can handle only locally-monotonic models and Trappist can handle general models, it is difficult to further compare between them. Hence, we focus on only comparisons between PyBoolNet and Trappist in the following observations.

The second observation is that the proposed method vastly outperforms PyBoolNet in computational time, on each and every model, and sometimes with orders of magnitude of difference (e.g., for most models in the 100–1000 nodes size range). Note that for all the cases where PyBoolNet did not manage to finish

Table 2: Timing comparisons between PyBoolNet (PBN), mpbn and Trappist on selected models from the literature. Column n (resp. s) denotes the number of nodes (resp. source nodes) of each model. Column $|M|$ denotes the number of minimal trap spaces and for each method is given the computation time in seconds. "DNF" means that the method did not finish the computation (stopping at the first 1000 minimal trap spaces, or all for the last column) within the timeout of two minutes. "NM" indicates a non-locally-monotonic model.

					PBN	mpbn	Trappist	
	model	n	s	$\|M\|$	1000	1000	1000	all
1	inflammatory-bowel [22]	47	0	1	DNF	NM	0.82	0.69
2	T-LGL-survival [22]	61	7	318	0.84	NM	0.01	0.01
3	butanol-production [22]	66	13	8192	0.71	NM	0.02	0.02
4	colon-cancer [22]	70	1	10	0.20	NM	0.01	0.01
5	mast-cell-activation [1]	73	19	>1000	0.76	NM	0.01	DNF
6	IL-6-signalling [22]	86	15	32768	1.29	NM	0.01	0.01
7	Corral-ThIL-17-diff [13]	92	16	>1000	DNF	NM	0.06	DNF
8	Korkut-2015 [30]	99	12	18556	DNF	1.50	0.07	0.08
9	adhesion-cip-migration [19]	121	4	78	36.80	0.35	0.08	0.09
10	interferon-1 [40]	121	55	>1000	10.11	NM	0.02	DNF
11	TCR-TLR5-signaling [44]	130	5	48	2.06	NM	0.03	0.02
12	influenza-replication [22]	131	11	10128	46.67	NM	0.03	0.02
13	prostate-cancer [32]	133	11	2760	DNF	NM	0.05	0.04
14	HIV-1 [22]	138	14	39424	DNF	NM	0.03	0.06
15	fibroblasts [21]	139	9	>1000	DNF	NM	0.06	DNF
16	HMOX-1-pathway [40]	145	56	>1000	6.52	NM	0.02	DNF
17	kynurenine-pathway [40]	150	72	>1000	DNF	NM	0.11	DNF
18	virus-replication-cycle [40]	154	25	>1000	DNF	NM	0.19	DNF
19	immune-system [22]	164	13	>1000	DNF	NM	0.11	DNF
20	RA-apoptosis [1]	180	59	>1000	DNF	NM	0.05	DNF
21	MAPK [1]	181	37	>1000	87.13	NM	0.04	DNF
22	er-stress [40]	182	75	>1000	17.88	NM	0.03	DNF
23	cascade-3 [49]	183	0	1	101.18	NM	0.40	0.07
24	CHO-2016 [30]	200	13	13312	DNF	3.19	0.06	0.08
25	T-cell-check-point [23]	218	14	>1000	69.94	NM	0.05	DNF
26	ErbB-receptor-signaling [20]	247	22	>1000	DNF	NM	0.23	DNF
27	macrophage-activation [22]	321	19	>1000	16.43	NM	0.04	DNF
28	cholocystokinin [1]	383	74	>1000	1.42	NM	0.14	DNF
29	Alzheimer [1]	762	237	>1000	DNF	NM	0.27	DNF
30	KEGG-network [29]	1659	521	>1000	DNF	23.44	3.60	DNF
31	human-network [25]	1953	669	>1000	DNF	25.63	5.77	DNF
32	SN-5 [26]	2746	829	>1000	DNF	33.57	DNF	DNF
33	turei-2016 [30]	4691	1257	???	DNF	DNF	DNF	DNF

before the timeout, as marked by "DNF" in Table 2, the timeout occurred during

the computation of the prime-implicants. Hence, not even a single minimal trap space was output by that method. The computational advantage is therefore immediately a practical advantage since on the one hand the state-of-the-art method did not allow any analysis whatsoever of the models, and on the other hand the proposed method could provide, very often under one second, the first thousand minimal trap spaces. For modellers having a critical look at a model and in a *model, invalidate, refine* loop this means a huge difference in the models that are amenable to study.

Note that even with a very restricted time-limit of two minutes, it was possible with the proposed technique to find *all* minimal trap spaces of small models (roughly under 130 nodes, i.e., considered as quite big up to now). Though it might seem impractical to handle tens of thousands of such possible complex attractors in a manual way, i.e., to compare them to specific experimental conditions and corresponding data, we hope that an automatic analysis of such attractors might become possible with systematic verification methods, not unlike that described in [23]. Since the ASP code generated is also quite declarative by nature, it is also possible to add to it supplementary constraints coming from the modeller in case one is looking for specific attractors. Finally, sampling from the ASP-generated solutions as is done in [12] would allow for a different type of exploration.

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

The last observation is that for very large models (i.e., more than two thousand nodes) the proposed method did not manage to finish the Petri net conversion before the timeout, as marked by "???" in Table 2. This points to the fact that our current choice of using a BDD-based translation to obtain that Petri net encoding, though it provides a small/efficient ASP might be too costly to handle the largest models. In such a case, a more *naive* encoding might provide a much larger ASP program, with many redundant rules, but easier/faster to obtain. The evaluation of the feasibility of such strategy, and of its impact on smaller instances, remains to be done and is out of the scope of this first article.

Note that though enumerating the extremal siphons of a Petri net is exponential (see [35] for instance) this is apparently not the bottleneck of the proposed method, showing once again that networks obtained from biochemical models do have a specific structure.

7 Conclusion

In this article we proposed a new method for the computation of minimal trap spaces of Boolean models, based on a new concept called maximal conflict-free siphons. This method is evaluated on large models from the literature and shows that it can scale up much better than the state-of-the-art prime-implicants based techniques. We believe that this opens up the way to a much better analysis of large Boolean models, which is needed with the advent of automatic model-generation pipelines [40].

Though we benchmarked this new approach against the state-of-the-art tool PyBoolNet, there are many more evaluations that we plan to do in the future. First, the BioLQM platform that we are using is providing another implicant-based method using BDDs in http://colomoto.org/biolqm/doc/tools-trapspace.html and though we expect it to behave mostly like PyBoolNet, that remains to be checked. Note that this also raises the question of replacing altogether the BioLQM preprocessing step we use to obtain the Petri net encoding, since its use of BDDs might not be optimal for that step. The trade-off between a small Petri net, and hence small ASP, and the time it takes to compute it (in other words, the trade-off of allowing redundant constraints) has to be evaluated in depth.

Second, the experimental results shown in this paper mostly expose the differences caused by the prime-implicant phase of PyBoolNet. There is much more information required to distinctly study the practical differences between our ASP encoding and that of PyBoolNet, and notably their size/efficiency ratio. Hence, we plan to conduct experiments on more real-world models or maybe random models that can be randomly generated by using BoolNet [34]. Such experiments should include a time-course of the number of ASP solutions found for proper comparison of the ASP encodings.

In addition, there are possibly other methods for computing maximal conflict-free siphons in Petri nets, like SAT/MaxSAT approaches [35]. Although these approaches do not directly support the maximal conflict-free siphon computation now, we plan to investigate them in the future. They could replace our ASP program if they outperform it.

However, the current method appears to already perform very well even on the biggest models we have considered. We thus plan to propose its inclusion in the CoLoMoTo effort to make it readily available to modellers.

References

1. Aghamiri, S.S., Singh, V., Naldi, A., Helikar, T., Soliman, S., Niarakis, A., Xu, J.: Automated inference of Boolean models from molecular interaction maps using CaSQ. Bioinform. **36**(16), 4473–4482 (2020). https://doi.org/10.1093/bioinformatics/btaa484
2. Angeli, D., Leenheer, P.D., Sontag, E.: A Petri net approach to persistence analysis in chemical reaction networks. In: Biology and Control Theory: Current Challenges, pp. 181–216. Springer (2007). https://doi.org/10.1007/978-3-540-71988-5_9

3. Angeli, D., Leenheer, P.D., Sontag, E.D.: Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws. SIAM J. Appl. Math. **71**(1), 128–146 (2011). https://doi.org/10.1137/090779401
4. Blätke, M.A., Heiner, M., Marwan, W.: Biomodel engineering with Petri nets. In: Algebraic and Discrete Mathematical Methods for Modern Biology, pp. 141–192. Elsevier (2015). https://doi.org/https://doi.org/10.1016/B978-0-12-801213-0.00007-1
5. Chaouiya, C., Bérenguier, D., Keating, S.M., Naldi, A., et al.: SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. BMC Syst. Biol. **7**, 135 (2013). https://doi.org/10.1186/1752-0509-7-135
6. Chaouiya, C., Naldi, A., Remy, E., Thieffry, D.: Petri net representation of multivalued logical regulatory graphs. Nat. Comput. **10**(2), 727–750 (2011). https://doi.org/10.1007/s11047-010-9178-0
7. Chaouiya, C., Naldi, A., Thieffry, D.: Logical modelling of gene regulatory networks with GINsim. In: Bacterial Molecular Networks, pp. 463–479. Springer (2012). https://doi.org/10.1007/978-1-61779-361-5_23
8. Chaouiya, C., Remy, E., Ruet, P., Thieffry, D.: Qualitative modelling of genetic networks: From logical regulatory graphs to standard Petri nets. In: Cortadella, J., Reisig, W. (eds.) Applications and Theory of Petri Nets 2004, 25th International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3099, pp. 137–156. Springer (2004). https://doi.org/10.1007/978-3-540-27793-4_9
9. Chatain, T., Haar, S., Jezequel, L., Paulevé, L., Schwoon, S.: Characterization of reachable attractors using Petri net unfoldings. In: Mendes, P., Dada, J.O., Smallbone, K. (eds.) Computational Methods in Systems Biology - 12th International Conference, CMSB 2014, Manchester, UK, November 17-19, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8859, pp. 129–142. Springer (2014). https://doi.org/10.1007/978-3-319-12982-2_10
10. Chatain, T., Haar, S., Kolcák, J., Paulevé, L., Thakkar, A.: Concurrency in Boolean networks. Nat. Comput. **19**(1), 91–109 (2020). https://doi.org/10.1007/s11047-019-09748-4
11. Chevalier, S., Froidevaux, C., Paulevé, L., Zinovyev, A.Y.: Synthesis of Boolean networks from biological dynamical constraints using answer-set programming. In: 31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019. pp. 34–41. IEEE (2019). https://doi.org/10.1109/ICTAI.2019.00014
12. Chevalier, S., Noël, V., Calzone, L., Zinovyev, A.Y., Paulevé, L.: Synthesis and simulation of ensembles of Boolean networks for cell fate decision. In: Abate, A., Petrov, T., Wolf, V. (eds.) Computational Methods in Systems Biology - 18th International Conference, CMSB 2020, Konstanz, Germany, September 23-25, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12314, pp. 193–209. Springer (2020). https://doi.org/10.1007/978-3-030-60327-4_11
13. Corral-Jara, K.F., Chauvin, C., Abou-Jaoudé, W., Grandclaudon, M., Naldi, A., Soumelis, V., Thieffry, D.: Interplay between SMAD2 and STAT5A is a critical determinant of IL-17A/IL-17F differential expression. Mol. Biomed. **2**(1), 1–16 (2021). https://doi.org/10.1186/s43556-021-00034-3
14. Degrand, E., Fages, F., Soliman, S.: Graphical conditions for rate independence in chemical reaction networks. In: Abate, A., Petrov, T., Wolf, V. (eds.) Computational Methods in Systems Biology - 18th International Conference, CMSB 2020,

Konstanz, Germany, September 23-25, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12314, pp. 61–78. Springer (2020). https://doi.org/10.1007/978-3-030-60327-4_4
15. Didier, G., Remy, E., Chaouiya, C.: Mapping multivalued onto Boolean dynamics. J. Theor. Biol. **270**(1), 177–184 (Feb 2011). https://doi.org/10.1016/j.jtbi.2010.09.017
16. Fontanals, L.C., Tonello, E., Siebert, H.: Control strategy identification via trap spaces in Boolean networks. In: Abate, A., Petrov, T., Wolf, V. (eds.) Computational Methods in Systems Biology - 18th International Conference, CMSB 2020, Konstanz, Germany, September 23-25, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12314, pp. 159–175. Springer (2020). https://doi.org/10.1007/978-3-030-60327-4_9
17. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011). https://doi.org/10.3233/AIC-2011-0491
18. Glass, L., Kauffman, S.A.: The logical analysis of continuous, non-linear biochemical control networks. J. Theor. Biol. **39**(1), 103–129 (1973). https://doi.org/10.1016/0022-5193(73)90208-7
19. Guberman, E., Sherief, H., Regan, E.R.: Boolean model of anchorage dependence and contact inhibition points to coordinated inhibition but semi-independent induction of proliferation and migration. Comput. Struct. Biotechnol. J. **18**, 2145–2165 (2020). https://doi.org/10.1016/j.csbj.2020.07.016
20. Helikar, T., Kochi, N., Kowal, B., Dimri, M., Naramura, M., Raja, S.M., Band, V., Band, H., Rogers, J.A.: A comprehensive, multi-scale dynamical model of ErbB receptor signal transduction in human mammary epithelial cells. PloS One **8**(4), e61757 (2013). https://doi.org/10.1371/journal.pone.0061757
21. Helikar, T., Konvalina, J., Heidel, J., Rogers, J.A.: Emergent decision-making in biological signal transduction networks. Proceedings of the National Academy of Sciences **105**(6), 1913–1918 (2008). https://doi.org/10.1073/pnas.0705088105
22. Helikar, T., Kowal, B.M., McClenathan, S., Bruckner, M., et al.: The Cell Collective: Toward an open and collaborative approach to systems biology. BMC Syst. Biol. **6**, 96 (2012). https://doi.org/10.1186/1752-0509-6-96
23. Hernandez, C., Thomas-Chollier, M., Naldi, A., Thieffry, D.: Computational verification of large logical models—application to the prediction of T cell response to checkpoint inhibitors. Front. Physiol. p. 1154 (2020). https://doi.org/10.3389/fphys.2020.558606
24. Keating, S.M., Waltemath, D., König, M., Zhang, F., et al.: SBML Level 3: an extensible format for the exchange and reuse of biological models. Mol Syst Biol. **16**(8), e9110 (2020). https://doi.org/10.15252/msb.20199110
25. Kim, J.R., Kim, J., Kwon, Y.K., Lee, H.Y., Heslop-Harrison, P., Cho, K.H.: Reduction of complex signaling networks to a representative kernel. Sci. Signal. **4**(175), ra35–ra35 (2011). https://doi.org/10.1126/scisignal.2001390
26. Kim, J., Yi, G.S.: RMOD: A tool for regulatory motif detection in signaling network. PloS One **8**(7), e68407 (2013). https://doi.org/10.1371/journal.pone.0068407
27. Klarner, H., Bockmayr, A., Siebert, H.: Computing maximal and minimal trap spaces of Boolean networks. Nat. Comput. **14**(4), 535–544 (2015). https://doi.org/10.1007/s11047-015-9520-7
28. Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. Bioinform. **33**(5), 770–772 (2017). https://doi.org/10.1093/bioinformatics/btw682

29. Kwon, Y.: Properties of Boolean dynamics by node classification using feedback loops in a network. BMC Syst. Biol. **10**, 83 (2016). https://doi.org/10.1186/s12918-016-0322-z
30. Lee, D., Cho, K.H.: Signal flow control of complex signaling networks. Sci. Rep. **9**(1), 1–18 (2019). https://doi.org/10.1038/s41598-019-50790-0
31. Liu, G., Barkaoui, K.: A survey of siphons in Petri nets. Inf. Sci. **363**, 198–220 (2016). https://doi.org/10.1016/j.ins.2015.08.037
32. Montagud, A., Béal, J., Tobalina, L., Traynard, P., Subramanian, V., Szalai, B., Alföldi, R., Puskás, L., Valencia, A., Barillot, E., Saez-Rodriguez, J., Calzone, L.: Patient-specific Boolean models of signaling networks guide personalized treatments. bioRxiv (Jul 2021). https://doi.org/10.1101/2021.07.28.454126
33. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (Apr 1989). https://doi.org/10.1109/5.24143
34. Müssel, C., Hopfensitz, M., Kestler, H.A.: BoolNet - an R package for generation, reconstruction and analysis of Boolean networks. Bioinform. **26**(10), 1378–1380 (2010). https://doi.org/10.1093/bioinformatics/btq124
35. Nabli, F., Martinez, T., Fages, F., Soliman, S.: On enumerating minimal siphons in Petri nets using CLP and SAT solvers: theoretical and practical complexity. Constraints An Int. J. **21**(2), 251–276 (2016). https://doi.org/10.1007/s10601-015-9190-1
36. Naldi, A., Monteiro, P.T., Müssel, C., for Logical Models, C., Tools, Kestler, H.A., Thieffry, D., Xenarios, I., Saez-Rodriguez, J., Helikar, T., Chaouiya, C.: Cooperative development of logical modelling standards and tools with CoLoMoTo. Bioinform. **31**(7), 1154–1159 (2015). https://doi.org/10.1093/bioinformatics/btv013
37. Noual, M., Regnault, D., Sené, S.: About non-monotony in Boolean automata networks. Theor. Comput. Sci. **504**, 12–25 (2013). https://doi.org/10.1016/j.tcs.2012.05.034
38. Oanea, O., Wimmel, H., Wolf, K.: New algorithms for deciding the siphon-trap property. In: Lilius, J., Penczek, W. (eds.) Applications and Theory of Petri Nets, 31st International Conference, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6128, pp. 267–286. Springer (2010). https://doi.org/10.1007/978-3-642-13675-7_16
39. Ogishima, S., Mizuno, S., Kikuchi, M., Miyashita, A., Kuwano, R., Tanaka, H., Nakaya, J.: AlzPathway, an updated map of curated signaling pathways: towards deciphering Alzheimer's disease pathogenesis. In: Systems Biology of Alzheimer's Disease, pp. 423–432. Springer (2016). https://doi.org/10.1007/978-1-4939-2627-5_25
40. Ostaszewski, M., Niarakis, A., Mazein, A., Kuperstein, I., Phair, R., Orta-Resendiz, A., Singh, V., Aghamiri, S.S., Acencio, M.L., Glaab, E., et al.: COVID19 Disease Map, a computational knowledge repository of virus–host interaction mechanisms. Mol Syst Biol. **17**(10), e10387 (2021). https://doi.org/10.15252/msb.202110387
41. Paulevé, L., Kolčák, J., Chatain, T., Haar, S.: Reconciling qualitative, abstract, and scalable modeling of biological networks. Nat. Commun. **11**(1) (Aug 2020). https://doi.org/10.1038/s41467-020-18112-5
42. Peterson, J.L.: Petri net theory and the modeling of systems. Prentice Hall PTR (1981)
43. Reddy, V.N., Mavrovouniotis, M.L., Liebman, M.N.: Petri net representations in metabolic pathways. In: Hunter, L., Searls, D.B., Shavlik, J.W. (eds.) Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, Bethesda, MD, USA, July 1993. pp. 328–336. AAAI (1993), http://www.aaai.org/Library/ISMB/1993/ismb93-038.php

44. Rodríguez-Jorge, O., Kempis-Calanis, L.A., Abou-Jaoudé, W., Gutiérrez-Reyna, D.Y., Hernandez, C., Ramirez-Pliego, O., Thomas-Chollier, M., Spicuglia, S., Santana, M.A., Thieffry, D.: Cooperation between T cell receptor and Toll-like receptor 5 signaling for CD4+ T cell activation. Sci. Signal. **12**(577), eaar3641 (2019). https://doi.org/10.1126/scisignal.aar3641
45. Singh, V., Ostaszewski, M., Kalliolias, G.D., Chiocchia, G., Olaso, R., Petit-Teixeira, E., Helikar, T., Niarakis, A.: Computational systems biology approach for the study of rheumatoid arthritis: from a molecular map to a dynamical model. Genom. Comput. Biol. **4**(1) (2018). https://doi.org/10.18547/gcb.2018.vol4.iss1.e100050
46. Thomas, R.: Boolean formalisation of genetic control circuits. J. Theor. Biol. **42**, 565–583 (1973). https://doi.org/10.1016/0022-5193(73)90247-6
47. Thomas, R.: Regulatory networks seen as asynchronous automata: a logical description. J. Theor. Biol. **153**(1), 1–23 (1991). https://doi.org/10.1016/S0022-5193(05)80350-9
48. Thomas, R., d'Ari, R.: Biological feedback. CRC press (1990)
49. Tsirvouli, E., Touré, V., Niederdorfer, B., Vázquez, M., Flobak, Å., Kuiper, M.: A middle-out modeling strategy to extend a colon cancer logical model improves drug synergy predictions in epithelial-derived cancer cell lines. Front. Mol. Biosci. **7** (Oct 2020). https://doi.org/10.3389/fmolb.2020.502573
50. Wang, R.S., Saadatpour, A., Albert, R.: Boolean modeling in systems biology: an overview of methodology and applications. Phys. Biol. **9**(5), 055001 (2012). https://doi.org/10.1088/1478-3975/9/5/055001
51. Zevedei-Oancea, I., Schuster, S.: Topological analysis of metabolic networks based on Petri net theory. Silico Biol. **3**(3), 323–345 (2003), http://content.iospress.com/articles/in-silico-biology/isb00100