# Concurrency in Boolean networks

Thomas Chatain, Stefan Haar, Juraj Kolčák, Loïc Paulevé, Aalok Thakkar

**Noname manuscript No.**
(will be inserted by the editor)

# Concurrency in Boolean networks

**Thomas Chatain · Stefan Haar · Juraj Kolčák · Loïc Paulevé · Aalok Thakkar**

Received: date / Accepted: date

**Abstract** Boolean networks (BNs) are widely used to model the qualitative dynamics of biological systems. Besides the logical rules determining the evolution of each component with respect to the state of its regulators, the scheduling of component updates can have a dramatic impact on the predicted behaviours. In this paper, we explore the use of Read (contextual) Petri Nets (RPNs) to study dynamics of BNs from a concurrency theory perspective. After showing bi-directional translations between RPNs and BNs and analogies between results on synchronism sensitivity, we illustrate that usual updating modes for BNs can miss plausible behaviours, i.e., incorrectly conclude on the absence/impossibility of reaching specific configurations. We propose an encoding of BNs capitalizing on the RPN semantics enabling more behaviour than the generalized asynchronous updating mode. The proposed encoding ensures a correct abstraction of any multivalued refinement, as one may ex-

T. Chatain, S. Haar, J. Kolčák, A. Thakkar
Inria Saclay-Île-de-France
LSV, CNRS, & ENS Paris-Saclay
Université Paris-Saclay, France
E-mail: thomas.chatain@lsv.fr
E-mail: stefan.haar@lsv.fr
E-mail: juraj.kolcak@lsv.fr
*Present address:* of A. Thakkar
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
E-mail: athakkar@seas.upenn.edu

L. Paulevé
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800
F-33400 Talence, France
CNRS, LRI UMR 8623, Univ. Paris-Sud – CNRS
Université Paris-Saclay, France
E-mail: loic.pauleve@labri.fr

pect to achieve when modelling biological systems with no assumption on its time features.

**Keywords** Discrete dynamical systems · Models of concurrency · Synchronism · Reachability

## 1 Introduction

Boolean networks (BNs) model dynamics of systems where several components (nodes) interact. They specify for each node an update function to determine its next value according to the configuration (global state) of the network. In addition, an *update mode* for scheduling the application of functions has to be specified to determine the set of reachable configurations.

BNs are increasingly used to model dynamics of biological interaction networks, such as gene networks and cellular signalling pathways. In these practical applications, it is usual to assess the accordance of a BN with the concrete modeled system by checking if the observed behaviours are reproducible by the abstract BN [37,43,18]. For instance, if one observes that the system can reach a configuration $y$ from configuration $x$, one may expect it is indeed the case in the BN model. The designed Boolean functions typically do not model the system correctly whenever it is not the case and should thus be fixed prior to further analysis. With this perspective, the choice of the update mode is crucial, as it is known to have a strong influence on the reachable configurations of the network.

More fundamentally, the relationships between different updating modes have been extensively studied for function-centered models such as cellular automata [39,5] and Boolean networks [26,42,21,3,30,31], on which this article is focused.

Interestingly, the study of updating mechanisms in networks and their effect on the emerging global dynamics has also been widely addressed in the field of discrete and hybrid concurrent systems, especially with Petri nets [24,8, 9,45,46]. Petri nets are a classical formal framework for studying concurrency, offering a fine-grained specification of the *conditions* (partial configurations) for *events* (partial configuration changes). This decomposed view of causality and effect of updates enables capturing events which can indifferently occur sequentially or in parallel, and events having conflicts (triggering one would pre-empt the application of the second).

In the literature, many variants of Petri nets have been employed to model and simulate various biological processes (see [22,33] for examples and [10] for a review paper), but little work considered the link between the theoretical work on concurrency in Petri nets and the theoretical work in Boolean networks. In [40,11,13], encodings of BNs and their multi-valued extension in certain classes of Petri nets have been proposed, often as means to take advantage of existing dynamical analysis already implemented for Petri nets, e.g., model-checking.

This paper aims at building a bridge between the theoretical work in BNs on the one hand and Read Petri Nets (RPNs), also known as contextual Petri

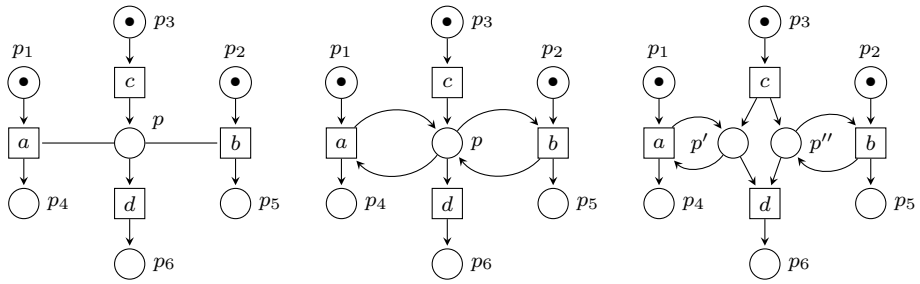


**Fig. 1** A Read Petri net $\mathcal{R}$ (left) and two different interpretations $\mathcal{N}_1$ and $\mathcal{N}_2$ (center, right) of $\mathcal{R}$ as ordinary Petri nets; following [7].

nets, on the other. RPNs augment ordinary Petri Nets (PNs) with read arcs to model read-only access to resources. It is always possible to simulate a RPN by an ordinary PN, see Figure 1 and the discussion below. Our choice of using RPN is motivated by the fact that the connection between BNs and RPNs is more intuitive; but there is also an important technical advantage in using RPNs directly, rather than equivalent ordinary models.

Let us examine Figure 1 more closely. In the read net $\mathcal{R}$ on the left hand side, transitions $a$ and $b$ will be enabled while $p$ is marked, i.e. between the firings of $c$ and $d$; once a token is available on $p$, both $a$ and $b$ can fire independently and jointly, because the read arcs linking them to $p$ do not require *removal* of the token from $p$. In $\mathcal{N}_1$ in the middle, the firing of $a$ and $b$ in any order is still possible, however their *synchronous* firing is prohibited by the conflict over the token on $p$. Only Petri net $\mathcal{N}_2$ is equivalent to $\mathcal{R}$; synchronous firing of $a$ and $b$ is obtained at the expense of duplicating $p$ by creating $p'$ and $p''$. In other words, faithful rendering of read net behaviour by ordinary nets requires the creation of multiple places for each place 'read' by several transitions, in order to pass from a read net to an ordinary net model. As will be seen, the constructions required for translation between read nets and BN in their turn also multiply place elements ; putting these constructions together is possible, but makes the resulting nets still larger and a lot less intuitive to apprehend and analyze.

In this paper, we consider the class of *safe* (or 1-bounded) RPNs where each place can be marked by at most one token, which makes it a natural choice for linking with Boolean networks. This class has been extensively studied in the literature and enables fine-grained definitions of different concurrent semantics as it is detailed in Sect. 3 and on which results of this article are built upon.

Below, we will give bi-directional equivalent connections between the two formalisms of BNs and RPNs; this allows to we use a classical result from Petri net theory to show the PSPACE-completeness of reachability in asynchronous BNs. Then, we exhibit analogies of results on update mode comparisons. Importantly, we show how the concurrent view of updates brings new updating modes for BNs, enabling new behaviours and meeting with a correct abstrac-

tion of multi-level systems. This result is illustrated on a small BN which occurs in different models of actual biological networks, and for which the usual updating modes fail to capture behaviours existing in refined models (Sect. 7.1; Fig. 9).

*Outline.* Sect. 2 gives basic definitions of BNs, their asynchronous, synchronous, and generalized asynchronous update mode, and their influence graph. Sect. 3 defines safe RPNs and their atomic, step, and interval semantics. Sect. 4 brings encodings of BNs into safe RPNs and vice-versa, the latter allowing to derive that reachability in BNs is PSPACE-complete. Sect. 5 establishes an analogy between the results on synchronism sensitivity in BNs and RPNs. Sect. 6 provides an encoding of the interval semantics of RPNs into asynchronous BNs, initially published in the conference paper [15]. Sect. 7 first illustrates the benefits of the interval semantics on a simple BN showing that usual BN semantics can miss plausible behaviours. Then, an extension of the interval semantics is proposed in order to meet with a correct abstraction of behaviours achievable in a multivalued refinement. Finally, Sect. 8 summarizes the contributions and discusses further work.

*Notations.* If $S$ is a finite set, $|S|$ denotes its cardinality. $\mathbb{B} = \{0, 1\}$, and we write $\wedge$, $\vee$, $\neg$ for logic operators *and*, *or*, *not*; given a set of literals $L = \{l_1, \ldots, l_k\}$, $\bigwedge_L \equiv l_1 \wedge \cdots \wedge l_k$ with $\bigwedge_\emptyset = 1$, and $\bigvee_L \equiv l_1 \vee \cdots \vee l_k$ with $\bigvee_\emptyset = 0$.

## 2 Boolean networks with function-centered specification

Given a *configuration* $x \in \mathbb{B}^n$ and $i \in \{1, \ldots, n\}$, we denote $x_i$ the $i^{\text{th}}$ component of $x$, so that $x = x_1 \ldots x_n$. Given two configurations $x, y \in \mathbb{B}^n$, the components that differ are noted $\Delta(x, y) \stackrel{\Delta}{=} \{i \in \{1, \ldots, n\} \mid x_i \neq y_i\}$.

**Definition 1 (Boolean network)** A Boolean network (BN) of dimension $n$ is a collection of functions $f = \langle f_1, \ldots, f_n \rangle$ where $\forall i \in \{1, \ldots, n\}, f_i : \mathbb{B}^n \to \mathbb{B}$.

Given $x \in \mathbb{B}^n$, we write $f(x)$ for $f_1(x) \ldots f_n(x)$.

Fig. 2 (a) shows an example of BN of dimension 3.

When modelling biological systems, each node $i \in \{1, \ldots, n\}$ usually represents a biochemical species, being either active (or present, value 1) or inactive (or absent, value 0). Each function $f_i$ indicates how is the evolution of the value of $i$ influenced by the current value of other components $j \in \{1, \ldots, n\}$. However, this description can be interpreted in several ways, therefore several updating modes coexist for BNs, depending on the assumptions about the order in which the evolutions predicted by the $f_i$ apply.

The (fully) *asynchronous updating* assumes that only one component is updated at each time step. The choice of the component to update is nondeterministic.

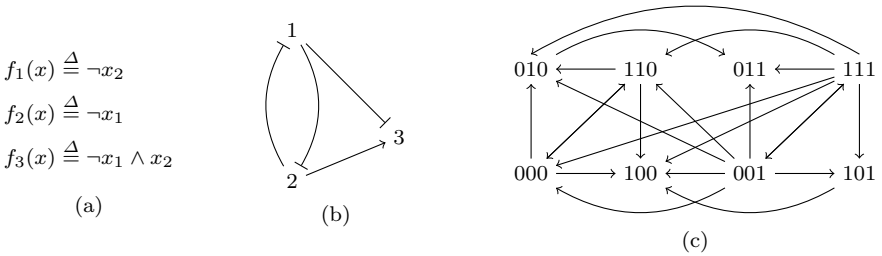


**Fig. 2** (a) Example BN $f$ of dimension 3; (b) Influence graph $G(f)$; positive edges are with normal tip; negative edges are with bar tip; (c) Transition relations between states in $\mathbb{B}^n$ according to the generalized asynchronous semantics of $f$.

**Definition 2 (Asynchronous updating)** Given a BN $f$, the binary irreflexive relation $\xrightarrow[\text{async}]{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:

$$x \xrightarrow[\text{async}]{f} y \overset{\Delta}{\iff} \exists i \in \{1, \ldots, n\}, \Delta(x, y) = \{i\} \wedge y_i = f_i(x) \ .$$

We write $\xrightarrow[\text{async}]{f}{}^*$ for the transitive closure of $\xrightarrow[\text{async}]{f}$.

The *synchronous updating* can be seen as the opposite: *all* components are updated at each time step. This leads to a purely deterministic dynamics.

**Definition 3 (Synchronous updating)** Given a BN $f$, the binary irreflexive relation $\xrightarrow[\text{sync}]{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:

$$x \xrightarrow[\text{sync}]{f} y \overset{\Delta}{\iff} x \neq y \wedge \forall i \in \{1, \ldots, n\}, y_i = f_i(x) \ .$$

By forcing all the components to evolve simultaneously, the synchronous updating makes a strong assumption on the dynamics of the system. In many concrete cases, for instance in systems biology, this assumption is often unrealistic, at least because the components model the quantity of some biochemical species which evolve at different speeds.

As a result, the synchronous updating fails to describe some behaviours, like the transition 010 $\to$ 011 represented in Fig. 2 (c) which represents the activation of species 3 when species 1 is inactive and species 2 is active ($f_3(010) = 1$). There are also transitions which are possible in the synchronous but not in the asynchronous updating, for instance 000 $\to$ 110. Remark that 110 is not even reachable from 000 in the asynchronous updating.

The *generalized asynchronous updating* generalizes both the asynchronous and the synchronous updating: it allows updating synchronously any nonempty subset of components.

**Definition 4 (Generalized asynchronous updating)** Given a BN $f$, the binary irreflexive relation $\xrightarrow[\text{gen}]{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:

$$x \xrightarrow[\text{gen}]{f} y \overset{\Delta}{\iff} x \neq y \wedge \forall i \in \Delta(x,y) : y_i = f_i(x) \ .$$

Clearly, $x \xrightarrow[\text{async}]{f} y \Rightarrow x \xrightarrow[\text{gen}]{f} y$ and $x \xrightarrow[\text{sync}]{f} y \Rightarrow x \xrightarrow[\text{gen}]{f} y$. The converse propositions are false in general.
Note that we forbid "idle" transitions ($x \to x$) regardless of the updating mode.

Other updating modes like sequential or block sequential have also been considered in the literature on cellular automata and BNs [5,3], and usually lead to transitions allowed by the generalized asynchronous updating.

For each node $i \in \{1,\dots,n\}$ of the BN, $f_i$ typically depends only on a subset of nodes of the network. The *influence graph* of a BN (also called interaction or causal graph) summarizes these dependencies by having an edge from node $j$ to $i$ if $f_i$ depends on the value of $j$. Formally, $f_i$ depends on $x_j$ if there exists a configuration $x \in \mathbb{B}^n$ such that $f_i(x)$ is different from $f_i(x')$ where $x'$ differs from $x$ solely in the component $j$ ($x'_j = \neg x_j$). Moreover, assuming $x_j = 0$ (therefore $x'_j = 1$), we say that $j$ has a positive influence on $i$ (in configuration $x$) if $f_i(x) < f_i(x')$, and a negative influence if $f_i(x) > f_i(x')$. It is possible that a node has different signs of influence on $i$ in different configurations, leading to non-monotonic $f_i$. It is worth noticing that different BNs can have the same influence graph.

**Definition 5 (Influence graph)** Given a BN $f$, its *influence graph* $G(f)$ is a directed graph $(\{1,\dots,n\}, E_+, E_-)$ with *positive* and *negative* edges such that

$$(j,i) \in E_+ \overset{\Delta}{\iff} \exists x,y \in \mathbb{B}^n : \Delta(x,y) = \{j\}, x_j < y_j, f_i(x) < f_i(y)$$
$$(j,i) \in E_- \overset{\Delta}{\iff} \exists x,y \in \mathbb{B}^n : \Delta(x,y) = \{j\}, x_j < y_j, f_i(x) > f_i(y)$$

A (directed) cycle composed of edges in $E_+ \cup E_-$ is said *positive* when it is composed by an even number of edges in $E_-$ (and any number of edges in $E_+$), otherwise it is *negative*.

When $E_+ \cap E_- = \emptyset$, we say that $f$ is *locally monotonic*.

The influence graph is an important object in the literature of BNs [41,2]. For instance, many studies have shown that one can derive dynamical features of a BN $f$ by the sole analysis of its influence graph $G(f)$. Importantly, the presence of negative and positive cycles in the influence graph, and the way they are intertwined can help to determine the nature of attractors (that are the smallest sets of configurations closed by the transition relationship) [35], and derive bounds on the number of fixpoints and attractors a BN having the same influence graph can have [34,1,4].

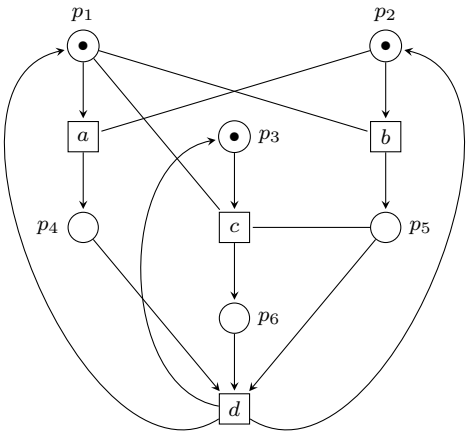


**Fig. 3** A Read Petri net (RPN). Neither atomic semantics nor step semantics allow $d$ to fire, while the more permissive non-atomic semantics allows it.

## 3 Read Petri Nets with transition-centered specifications

In the semantics of BNs, each node computes its next value according to the value of the other nodes. We have seen in the previous section that this general rule does not suffice to define the precise behaviour and several updating modes can be considered.

This situation is very similar to what happens in contextual or Read Petri nets (RPNs), where read arcs have been introduced to model read-only access to resources, for a matter of concurrency. Interestingly, the introduction of read arcs in Petri nets has also led to several variants of the semantics. In this section, we present some of them, mainly taken from [14]. Next, relying on a natural encoding of BNs in RPNs (Sect. 4), we will establish a correspondence between updating modes for BNs and semantics of RPNs. In particular, we transpose the *interval semantics* of RPNs to a new semantics for BNs (Sect. 6) which retrieves some plausible scenarios that were missed by other updating modes.

### 3.1 Read Petri Nets

We consider only *safe* Read Petri nets (RPNs), i.e., RPNs with at most one token in each place at any time.

**Definition 6 (Read Petri Net (RPN))** A *Read Petri net* is a tuple $(P, T, pre, cont, post, M_0)$ where $P$ and $T$ are finite sets of *places* and *transitions* respectively, *pre*, *cont* and *post* map each transition $t \in T$ to its (nonempty) *preset* denoted ${}^\bullet t \stackrel{\Delta}{=} pre(t) \subseteq P$, its (possibly empty) *context*

denoted $\underline{t} \triangleq cont(t) \subseteq P \setminus {}^\bullet t$ and its (possibly empty) *postset* denoted $t^\bullet \triangleq post(t) \subseteq P$; $M_0 \subseteq P$ is the *initial marking*. We usually denote ${}^\bullet\underline{t} \triangleq {}^\bullet t \cup \underline{t}$.

For simplicity, we assume that for every transition $t$, its context is disjoint from its preset and postset.

A RPN is represented as a graph with two types of nodes: places (circles) and transitions (rectangles). Presets are represented by arrows from places to transitions, postsets by arrows from transitions to places, and contexts by undirected edges, called read arcs, between places and transitions. The initial marking is represented by tokens in places. Fig. 3 shows an example of RPN. The transition $a$, for instance, has $p_1$ in its preset, $p_2$ in its context and $p_4$ in its postset.

### 3.2 Atomic Semantics

A *marking* of a safe RPN is a set $M \subseteq P$ of marked places. A Petri net starts in its *initial marking* $M_0$. A transition $t \in T$ is *enabled* in a marking $M$ if all the places of its preset and context are marked, i.e., ${}^\bullet\underline{t} \subseteq M$. Then $t$ can *fire* from $M$, leading to the marking $M' \triangleq (M \setminus {}^\bullet t) \cup t^\bullet$. In this case, we write $M \xrightarrow[\text{atom}]{N,t} M'$ or simply $M \xrightarrow[\text{atom}]{N} M'$.

As we consider only *safe* RPNs, we assume that if a transition $t \in T$ is enabled in a marking $M$, then $(M \setminus {}^\bullet t) \cap t^\bullet = \emptyset$.

**Definition 7 (Atomic semantics, a-run)** We call *firing sequence of $N$ under the atomic semantics*, or *a-run*, any sequence $\sigma \triangleq (t_1 \dots t_n)$ of transitions for which there exist markings $M_1, \dots, M_n$ such that for all $i \in \{1, \dots, n\}$, firing $t_i$ from $M_{i-1}$ is possible and leads to $M_i$.

For instance, the net in Fig. 3 has two possible firing sequences: $(a)$ and $(bc)$. However, $d$ can never fire because that would require to fire both $a$ and $b$ first, and firing one of $a, b$ disables the other.

### 3.3 Non-atomic Semantics

In this section, we discuss two semantics for concurrent firing of multiple transitions. One is the well-known *step semantics* [23], in which multiple transitions can fire simultaneously. This is typically the case of $a$ and $b$ in the net of Fig. 3, which are both enabled and have disjoint presets, but cannot fire together according to the atomic semantics. The step semantics can be interpreted as first checking whether all members of a set of transitions can fire, and then firing them simultaneously. Intuitively, the step semantics is somehow similar to the general asynchronous updating as it considers any set of fireable transitions; whereas the maximal step semantics which considers only maximal sets of

fireable transitions is analoguous to the synchronous updating. We then recall the *interval semantics* introduced in [14], which allows a more liberal choice of checking and firing transitions in a set.

We present the semantics under the assumption that the underlying net is safe even under these two semantics, which allow more possibilities than the atomic one.

### *3.3.1 Step Semantics*

We first recall the step semantics [23].

**Definition 8 (Step semantics, s-run)** Let $N$ be a RPN. A *step* is a *set* $S$ of transitions of $N$. It can fire from configuration $M$ and lead to configuration $M'$, written $M \xrightarrow[\text{step}]{N,S} M'$ or simply $M \xrightarrow[\text{step}]{N} M'$, if

- every $t \in S$ is enabled in $M$,
- the presets of the transitions in $S$ are disjoint, and
- $M' = \big(M \setminus \bigcup_{t\in S} {}^\bullet t\big) \cup \bigcup_{t\in S} t^\bullet$.

We call *s-run* of $N$ any sequence $\sigma \stackrel{\Delta}{=} (S_1 \dots S_n)$ of *steps* for which there exist markings $M_1, \dots, M_n$ such that for all $i \in \{1, \dots, n\}$, step $S_i$ can fire from $M_{i-1}$ and leads to $M_i$.

A variant of step semantics, called maximal step semantics has received interest in the literature [25,19].

**Definition 9 (Maximal step semantics)** The firing rule for the maximal step semantics is defined as $M \xrightarrow[\text{mstep}]{N,S} M'$ (or simply $M \xrightarrow[\text{mstep}]{N} M'$) iff $M \xrightarrow[\text{step}]{N,S} M'$ and no larger step $S' \supsetneq S$ can fire from $M$.

In the example of Fig. 3, the step semantics allows one to fire $a$ and $b$ in one step since they are both enabled in the initial marking and ${}^\bullet a \cap {}^\bullet b = \emptyset$. This gives the s-run $(\{a, b\})$ in addition to the others which were already possible under the atomic semantics; for instance the a-run involving $b$ followed by $c$, denoted $(bc)$ for the atomic semantics, is simply rewritten as the s-run $(\{b\}\{c\})$ under the step semantics. However, transition $d$ remains dead since none of these s-runs contains all of $a$, $b$, and $c$.

The intuitive model underlying the step semantics is that all the transitions in the step can first check, in any order, whether they are enabled and not in conflict with one another, i.e., their presets are disjoint. Once the checks have been performed, they can all fire, again in any order. Put differently, if we denote the checking phase of a transition $t$ by $t^-$ and its firing phase by $t^+$, then every step consists of any permutation of the actions of type $t^-$ (for all transitions $t$ in the step), followed by any permutation of the actions $t^+$. The notion introduced in Def. 10 formalizes this intuition.

**Definition 10 ($s^{\pm}$-run)** For every s-run $(T_1 \dots T_n)$ of a RPN $N$, every concatenation $u_1^-.u_1^+.\cdots.u_n^-.u_n^+$ of sequences $u_i^-$ and $u_i^+$, is an *$s^{\pm}$-run* of $N$, where every $u_i^-$ is a permutation of the set $\{t^- \mid t \in T_i\}$ and every $u_i^+$ is a permutation of the set $\{t^+ \mid t \in T_i\}$ (where $T_i$ is a set of transitions of $N$).

For example, the s-run $(\{b\}\{c\})$ yields the $s^{\pm}$-run $(b^-b^+c^-c^+)$ and the s-run $(\{a,b\})$ yields four $s^{\pm}$-runs: $(a^-b^-a^+b^+)$, $(a^-b^-b^+a^+)$, $(b^-a^-a^+b^+)$ and $(b^-a^-b^+a^+)$.

### *3.3.2 Splitting Transitions for Understanding Steps*

Def. 10 formalizes a semantics of RPNs in which the firing of a transition does not happen directly, but in two steps, the checking of the pre-conditions and the actual execution. In this section, we generalize this idea.

The left-hand side of Fig. 4 shows a part of the net in Fig. 3, which consists of transition $a$ with its preset $\{p_1\}$, context $\{p_2\}$, and postset $\{p_4\}$. The construction on the right-hand side of 4 illustrates the idea of splitting firing transitions into two phases:

- every transition $t$ is split into $t^-$ and $t^+$;
- every place $p$ is duplicated to $p^c$ (meaning token in $p$ available for consumption) and $p^r$ (meaning token in $p$ available for reading).

Similar ideas about splitting transitions can be found in several works, for instance in [44].

Intuitively, if we apply this construction to all transitions from Fig. 3, then the $s^{\pm}$-runs of that net correspond to a-runs of the newly constructed net. The following Def. 11 provides the precise details of the construction.

**Definition 11 ($split(N)$)** Given a RPN $N = (P, T, pre, cont, post, M_0)$, $split(N) \triangleq (P', T', pre', cont', post', M_0')$ is the RPN where

- $T'$ contains two copies, denoted $t^-$ and $t^+$ of every transition $t \in T$.
- $P'$ contains two copies, denoted $p^c$ and $p^r$ of every place $p \in P$, plus one place $p_t$ per transition $t \in T$.
- ${}^\bullet t^- \triangleq \{p^c \mid p \in {}^\bullet t\}$
- $\underline{t^-} \triangleq \{p^r \mid p \in \underline{t}\}$
- $t^{-\bullet} \triangleq \{p_t\}$
- ${}^\bullet t^+ \triangleq \{p^r \mid p \in {}^\bullet t\} \cup \{p_t\}$
- $\underline{t^+} \triangleq \emptyset$
- $t^{+\bullet} \triangleq \{p^c \mid p \in t^\bullet\} \cup \{p^r \mid p \in t^\bullet\}\}$
- $M_0' \triangleq \{p^c \mid p \in M_0\} \cup \{p^r \mid p \in M_0\}$

We now formally prove the intuition mentioned above:

**Lemma 1** *Every $s^{\pm}$-run $\sigma^{\pm}$ of $N$ is an a-run of $split(N)$. Moreover $\sigma^{\pm}$ reaches the marking $\{p^c \mid p \in M\} \cup \{p^r \mid p \in M\}$, where $M$ is the marking of $N$ reached after the s-run $\sigma$ from which $\sigma^{\pm}$ is obtained.*

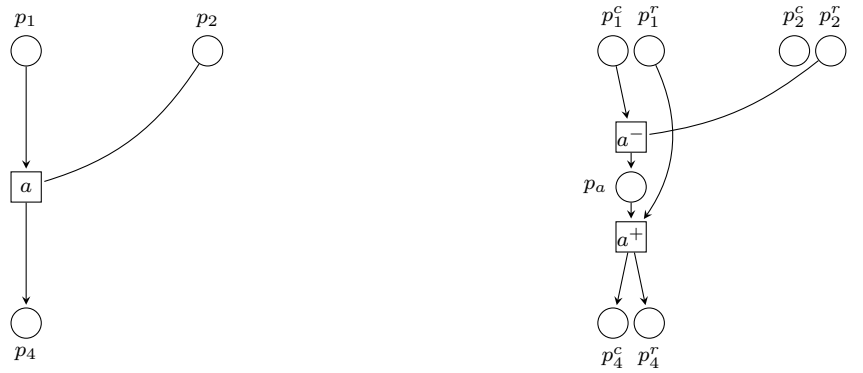


**Fig. 4** The splitting of transition $a$ (left) into $a^-$ and $a^+$ (right).

*Proof* We proceed by induction on the length of $\sigma$. The case $\sigma = ()$ is trivial. Now, let $\sigma^{\pm} = u_1^-.u_1^+.\cdots.u_n^-.u_n^+$ be an s$^{\pm}$-run obtained from an s-run $\sigma = (T_1 \dots T_n)$, assume the property true for $u_1^-.u_1^+.\cdots.u_{n-1}^-.u_{n-1}^+$ and denote $M_{n-1}$ the marking reached after $(T_1 \dots T_{n-1})$. By induction hypothesis, $u_1^-.u_1^+.\cdots.u_{n-1}^-.u_{n-1}^+$ reaches the marking $\{p^c \mid p \in M_{n-1}\} \cup \{p^r \mid p \in M_{n-1}\}$ of $split(N)$. The fact that $T_n$ is a valid step from $M_{n-1}$ implies that $\bigcup_{t \in T_n} {}^\bullet\underline{t} \subseteq M_{n-1}$ and that the presets of the transitions in $T_n$ are disjoint. This allows one to fire all the $t^-$, $t \in T_n$ in any order and reach the marking $\{p^c \mid p \in M_{n-1} \setminus \bigcup_{t \in T_n} {}^\bullet t\} \cup \{p^r \mid p \in M_{n-1}\} \cup \{p_t \mid t \in T_n\}$ of $split(N)$. Now the $t^+$, $t \in T_n$, are all enabled and their presets are disjoint. They can in turn be fired in any order, reaching the desired marking of $split(N)$. □

Note that the converse of Lemma 1 does not hold. For instance, for the net $N$ from Fig. 3, the net $split(N)$ admits the a-run $a^-b^-b^+c^-c^+a^+$, which is not an s$^{\pm}$-run of $N$.

### *3.3.3 Interval Semantics*

We have seen that the construction $split(N)$ admits firing sequences that cannot be mapped back to executions under either the atomic or the step semantics. In this section, we shall introduce the *interval semantics*, which is more general than the step semantics, and whose interpretation on a net $N$ does correspond to the feasible executions in $split(N)$.

**Definition 12 (Interval semantics, i-run)** Every a-run of $split(N)$ is called *i-run* of $N$, or run of $N$ under the interval semantics.

Coming back to the example of Fig. 3, transition $d$ can fire under the interval semantics, for instance after the i-run $a^-b^-b^+c^-c^+a^+d^-d^+$ where transitions $b$ and $c$ complete the firing during the period in which $a$ fires. Under the atomic semantics, $a$ and $b$ are in conflict, which prevents $d$ from firing. Under

the step semantics, $a$ and $b$ can fire in the same step, but then $c$ cannot fire. Under the interval semantics, $d$ can also fire.

Recall that we introduced $t^-$ and $t^+$ to represent different phases during the execution of transition $t$. An obvious question is whether the new semantics can lead to runs in which a transition 'gets stuck' during its execution. The following Lemma 2 affirms that this is not the case: once $t^-$ is fired, nothing can hinder $t^+$ from firing too.

**Definition 13 (complete marking)** A marking of $split(N)$ is *complete* if no $p_t$ is marked.

In particular, the initial marking is complete.

**Definition 14 (complete i-run)** An i-run is *complete* if for each transition $t^-$ in it, it includes the corresponding transition $t^+$.

**Lemma 2** *Every i-run can be completed: for every i-run $\sigma$, there exists a suffix $\mu$ which matches all the unmatched $t^-$, and such that $\sigma\mu$ is an i-run. Moreover, complete i-runs (and only them) lead to complete markings.*

*Proof* As long as a $t^-$ is unmatched, ${}^\bullet t^+$ remains included in the marking: no other transition consumes these tokens. Hence, it suffices to fire all the $t^+$ corresponding to the unmatched $t^-$, in any order. □

Now, relating $split(N)$ with the original net $N$, we map naturally every marking $M$ of $N$ to the complete marking $M'$ of $split(N)$ defined as $M' \stackrel{\Delta}{=} \{p^c \mid p \in M\} \cup \{p^r \mid p \in M\}$. We get of course that

$$M_1 \xrightarrow[\text{atom}]{N,t} M_2 \implies M_1' \xrightarrow[\text{atom}]{N,t^-} \xrightarrow[\text{atom}]{N,t^+} M_2' ,$$

but in general the interval semantics induces more runs: for all markings $M_1$ and $M_2$ of $N$, we write $M_1 \xrightarrow[\text{istep}]{N}{}^* M_2$ when $M_1' \xrightarrow[\text{atom}]{N}{}^* M_2'$.

## 4 Encodings

### 4.1 Coding Boolean Networks in safe Read Petri nets

The translation of BNs into safe Petri nets has been addressed in the literature (e.g. [10,11,13,12]). We provide here a similar encoding of BNs into safe RPNs, with the explicit specification of the context of transitions, and with notations that will be used in Sect. 5. The encoding can be easily generalized to multi-valued networks to safe RPNs, following [13,32].

BNs translate into a special type of RPNs:

– *complemented*: for every place $p$ there is exactly one distinct place $\overline{p}$ such that

$$ {}^\bullet p = \overline{p}^\bullet \ \wedge \ p^\bullet = {}^\bullet \overline{p} \ \wedge \ \forall t \in T : \ p \in \underline{t} \Rightarrow \overline{p} \notin \underline{t};$$

– *Boolean*: there is a surjection $var : P \to \{1, \ldots, n\}$ such that

$$\forall p, p' \in P : \ var(p) = var(p') \Leftrightarrow p' \in \{p, \overline{p}\},$$

and, subsequently, a mapping $val : P \to \mathbb{B}$ which satisfies

$$\forall p \in P : val(p) + val(\overline{p}) = 1.$$

Moreover, any reachable marking $M$ satisfies

$$\forall p \in P : p \in M \Leftrightarrow \overline{p} \notin M.$$

– *transition dichotomy*: every transition $t \in T$ has exactly one input place $p$ and one output place $\bar{p}$. If $val(p) = 0$ then call $t$ the *up-transition* $\mathbf{up}(var(p))$ of $var(p)$, otherwise the *down-transition* $\mathbf{dw}(var(p))$ of $var(p)$.

Let us consider a BN $f$ of dimension $n$. Each component $\mathsf{v} \in \{1, \ldots, n\}$ is modeled as two places $\mathsf{v}_0$ and $\mathsf{v}_1$ representing the two values possible for $\mathsf{v}$. Then a Petri net transition $\mathsf{v}^+$ is defined for each conjunctive clause of the disjunctive normal form of $(\neg x_\mathsf{v} \wedge f_\mathsf{v}(x))$. Such a transition consumes a token in the place $\mathsf{v}_0$ and produces a token in the place $\mathsf{v}_1$, and its context is formed by the places corresponding to the literals of the conjunction other than $\neg x_\mathsf{v}$: for each component $\mathsf{v}' \in \{1, \ldots, n\}$, $\mathsf{v}' \neq \mathsf{v}$, if the clause contains $x_{\mathsf{v}'}$, the context contains the place $\mathsf{v}'_1$; if the clause contains $\neg x_{\mathsf{v}'}$, the context contains the place $\mathsf{v}'_0$. A transition $\mathsf{v}^-$ is defined similarly for each conjunctive clause of the disjunctive normal form of $(x_\mathsf{v} \wedge \neg f_\mathsf{v}(x))$, such that

$${}^\bullet(\mathsf{v}^+) = (\mathsf{v}^-)^\bullet = \{\mathsf{v}_0\} \textit{ and } {}^\bullet(\mathsf{v}^-) = (\mathsf{v}^+)^\bullet = \{\mathsf{v}_1\},$$

Fig.5 shows the translation of the BN of Fig. 2 into RPN.

Hereafter, Def. 15 gives a formalization of this encoding, and Theorem 1 states its correctness with respect to the asynchronous, synchronous, generalized asynchronous updating modes, and RPN atomic, maximal step, and step semantics, respectively. Given a Boolean formula $F$, we write $\mathrm{DNF}[F]$ for the set of conjunctive clauses in the disjunctive normal form of $F'$. A clause $C \in \mathrm{DNF}[F]$ is then a set of literals, positives or negatives. It is worth noticing that the resulting RPN can have a number of transitions exponential in the number of literals in the Boolean functions.

**Definition 15** Given a BN $f$ of dimension $n$ and a configuration $y$, $(\!|f|\!)$ is the RPN $(P, T, pre, cont, post, M_0)$ such that

– $P = \{1, \ldots, 2n\}$ are the places;
– $T, pre, cont, post$ are the smallest sets such that for each $i \in \{1, \ldots, n\}$, for each clause $C \in \mathrm{DNF}[\neg x_i \wedge f_i(x)]$ (resp. $C \in \mathrm{DNF}[x_i \wedge \neg f_i(x)]$), there is a transition $t \in T$ such that ${}^\bullet t = \{i\}$ (resp. ${}^\bullet t = \{i + n\}$), $t^\bullet = \{i + n\}$ (resp. $t^\bullet = \{i\}$), and $\underline{t} = \{j \mid [\neg x_j] \in C, j \neq i\} \cup \{j + n \mid [x_j] \in C, j \neq i\}$;
– $M_0 = (\!|y|\!)$

where, for any configuration $x \in \mathbb{B}^n$, $(\!|x|\!) \stackrel{\Delta}{=} \{i + nx_i \mid i \in \{1, \ldots, n\}\}$ (e.g., $(\!|010|\!) = \{1, 5, 3\}$, $(\!|101|\!) = \{4, 2, 6\}$).

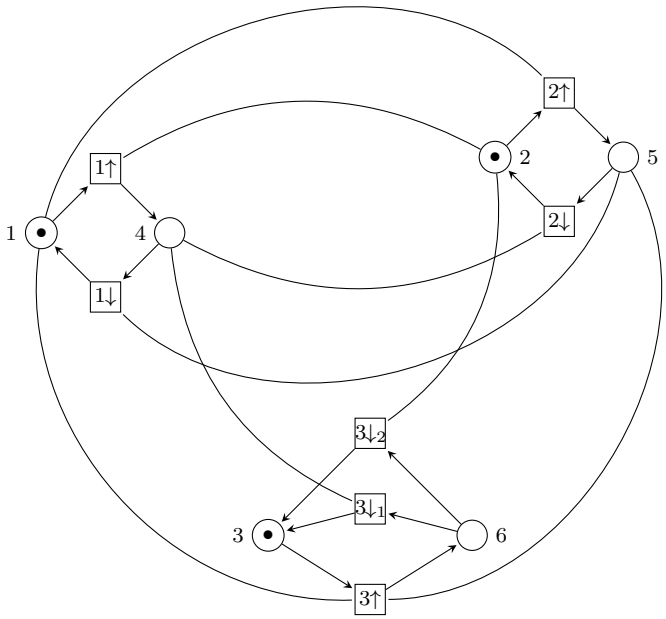


**Fig. 5** RPN encoding of the BN of Fig. 2 $\langle f_1(x) = \neg x_2, f_2(x) = \neg x_1, f_3(x) = \neg x_1 \wedge x_2 \rangle$ and configuration 000

**Theorem 1** *Given a BN $f$ of dimension $n$, for any configurations $x, y \in \mathbb{B}^n$,*

$$x \xrightarrow[\text{async}]{f} y \iff (\!|x|\!) \xrightarrow[\text{atom}]{(\!|f|\!)} (\!|y|\!) \ ,$$

$$x \xrightarrow[\text{sync}]{f} y \iff (\!|x|\!) \xrightarrow[\text{mstep}]{(\!|f|\!)} (\!|y|\!) \ ,$$

$$x \xrightarrow[\text{gen}]{f} y \iff (\!|x|\!) \xrightarrow[\text{step}]{(\!|f|\!)} (\!|y|\!) \ .$$

*Proof* For any $i \in \{1, \ldots, n\}$, $f_i(x) \neq x_i$ if and only if there exists a transition $t$ of $(\!|f|\!)$ where ${}^\bullet\underline{t} \subseteq (\!|x|\!)$. □

## 4.2 Coding Read Petri Nets in Boolean Networks

We have given above a translation of BNs into (a special class of) RPNs. The comparison of both models also leads us into the opposite direction.

In the following, fix a safe RPN $N = (P, T, pre, cont, post, M_0)$. The BN associated to $N$ has $|P| + |T|$ components, where the first $|P|$ components encode the marking of the corresponding places, and the $|T|$ other components encode the occurring transitions. Without loss of generality, we assume that

places and transitions range over indexes from 1 to $|P| + |T|$, i.e., $P \cup T \equiv \{1, \ldots, |P| + |T|\}$. In order to simplify the encodings, we additionally assume the RPNs to be *loop-free*, i.e., for every transition $t \in T$, ${}^\bullet t \cap t^\bullet = \emptyset$. It is well known that loops can be replaced by read arcs without any effect on the (atomic) semantics.

Transporting the *dynamics*, i.e., the actual *firing* of transitions, into the framework of BNs constitutes the non-trivial part of the translation. A RPN transition typically has more than one output place, while the functions in BNs write on one single variable. Our encoding decomposes the firing of a RPN transition into several updates of the BN. Essentially, when components corresponding to the pre-condition and context of a transition $t$ are marked, and if no other transition $t'$ is already occurring, the $t^{\text{th}}$ component of the BN can be updated to 1. Then, the components related to the input and output places of $t$ are updated (in any order) to apply their respective un-marking and marking. Once all these components have been updated, the $t^{\text{th}}$ component is updated to 0.

It results that a transition $t$ is occurring, encoded by the value 1 of the $t^{\text{th}}$ component, if and only if either (i) no transition is occurring, and all components corresponding to places in the pre-condition and context of $t$ have value 1, or (ii) $t$ is already occurring and at least one input (resp. output) place has not been unmarked (resp. marked) yet. A component corresponding to a place $p$ has value 1 if and only if either one of transition producing $p$ is occurring, or if it has already value 1 and none transition consuming it is occurring.

Hereafter, Def. 16 provides a formalization of the encoding of a safe RPN into a BN and Theorem 2 states its correctness in the scope of the asynchronous updating (atomic); note that the correctness also holds for the generalized asynchronous (step semantics) and synchronous (maximal step semantics) updating.

**Definition 16** Given a safe loop-free RPN $N = (P, T, pre, cont, post, M_0)$, $[\![N]\!]$ is the BN of dimension $|P| + |T|$ such that

$$\forall p \in P, [\![N]\!]_p(x) = \left( \bigvee_{t \in {}^\bullet p} x_t \right) \vee \left( x_p \wedge \bigwedge_{t \in p^\bullet} \neg x_t \right)$$

$$\forall t \in T, [\![N]\!]_t(x) = \left( \bigwedge_{p \in {}^\bullet \underline{t}} x_p \wedge \bigwedge_{t' \in T} \neg x_{t'} \right) \vee \left( x_t \wedge \left( \bigvee_{p \in t^\bullet} \neg x_p \vee \bigvee_{p \in {}^\bullet t} x_p \right) \right)$$

Given a marking $M \subseteq P$ of $N$, the corresponding configuration of $[\![N]\!]$ is $[\![M]\!] \in \mathbb{B}^n$ where $\forall p \in M, [\![M]\!]_p = 1$, $\forall p \in P \setminus M, [\![M]\!]_p = 0$, and $\forall t \in T, [\![M]\!]_t = 0$.

As an example, let us consider the RPN of Fig. 4(left), which consists 3 places $p_1, p_2, p_4$, and one transition $a$, such that ${}^\bullet a = \{p_1\}$, $a^\bullet = \{p_4\}$, and $\underline{a} = \{p_2\}$. The above encoding into BN leads to 4 Boolean functions:

$$f_{p_1}(x) = x_{p_1} \wedge \neg x_a \qquad f_{p_2}(x) = x_{p_2} \qquad f_{p_4}(x) = x_a \vee x_{p_4}$$
$$f_a(x) = (x_{p_1} \wedge x_{p_2}) \vee (x_a \wedge (\neg x_{p_4} \vee x_{p_1}))$$

**Theorem 2** *For a safe RPN $N = (P, T, pre, cont, post, M_0)$, and any pair of markings $M, M' \subseteq P$, one has*

$$M \xrightarrow[\text{atom}]{N}{}^* M' \iff [\![M]\!] \xrightarrow[\text{async}]{[\![N]\!]}{}^* [\![M']\!]$$

*Proof* If $M = M'$, the proof is trivial; in the following we consider $M \neq M'$.

($\Rightarrow$) Let us assume that $M \xrightarrow[\text{atom}]{N} M'$. Then there exists $t \in T$ such that ${}^\bullet\underline{t} \subseteq M$ and $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. Thus, $[\![N]\!]_t([\![M]\!]) = 1$, and therefore, there exists $y \in \mathbb{B}^n$ such that $x \xrightarrow[\text{async}]{[\![N]\!]} y$ with $\Delta(x, y) = \{t\}$. Then, assuming ${}^\bullet t \cap t^\bullet = \emptyset$, for each place $p \in {}^\bullet t$, because $t \in p^\bullet$ and $y_p = 1$, $[\![N]\!]_t(y) = 0$, and for each place $p \in t^\bullet$, because $t \in {}^\bullet p$, $[\![N]\!]_p = 1$. Therefore, by updating the components $p$ for $p \in {}^\bullet t \cup t^\bullet$ in any ordering, we obtain a configuration $z$ where all components are 0 except the components $p, \forall p \in M'$, and the component $t$. Then, because $[\![N]\!]_t(z) = 0$, the latter component is set to 0, resulting in the configuration $[\![M']\!]$.

($\Leftarrow$) Let us assume there exists $y \in \mathbb{B}^{|P|+|T|}$ such that $[\![M]\!] \xrightarrow[\text{async}]{[\![N]\!]} y$. Necessarily, there is a unique $t \in T$ such that $y_t = 1$; moreover, ${}^\bullet\underline{t} \subseteq M$. Remark that as long as the $t^{\text{th}}$ component of a configuration $x$ is 1, none of the other components $t'$ for $t' \in T, t' \neq t$ can be set to 1 (because $[\![N]\!]_{t'}(x) = 0$). Moreover, remark that in the configuration $y$, $\{p \in P \mid y_p \neq [\![N]\!]_p\} = {}^\bullet t \cup t^\bullet$, and that the component $t$ can be set to 0 only when all these latter components have been updated. Therefore, with $M'' = (M \setminus {}^\bullet t) \cup t^\bullet$, we obtain that $[\![M]\!] \xrightarrow[\text{async}]{[\![N]\!]}{}^* [\![M'']\!]$ and $M \xrightarrow[\text{atom}]{N} M''$. □

The reachability problem consists in deciding if there exists a sequence of transitions from a given configuration (marking) $x$ to a given configuration $y$. The reachability problem is PSPACE-complete in safe RPNs with asynchronous update mode [17]. By linear reduction to BNs, we therefore obtain that reachability in BNs is PSPACE-hard:

**Corollary 1** *Reachability in asynchronous BNs is PSPACE-hard.*

Finally, one can remark that deciding the reachability in BNs is in PSPACE: given a BN of dimension $n$ and the initial configuration $x$, let us define a counter using $n$ bits, initially with value 0. Then, while the counter has value strictly less that $2^n$ and the current configuration is not equal to $y$, non-deterministically apply an update, and increase the counter by one.

**Theorem 3** *Reachability in asynchronous BNs is PSPACE-complete.*

## 5 Synchronism sensitivity

For some BN or RPN, changing the update/firing policy (from synchronous to asynchronous) may have little impact on the reachable states. For others, it may render configurations reachable, or exclude previously feasible paths. We say that a network of the latter category is *synchronism sensitive*. The authors of [30] have analyzed this sensitivity in BNs; in this section, we perform an analogous analysis for RPNs. As we will show, the characterization of synchronism sensitivity in safe RPNs boils down to the existence of *preemption cycles*, defined below, among the transitions that are enabled in a given marking. Moreover, we show that when instantiated on RPNs encoding of BNs (according to Sect. 4.1), the general characterization of synchronism sensitivity in RPNs allows to recover the results of synchronism sensitivity in BNs with respect to their influence graph [30], with a slight generalization relaxing the local monotonicity constraints of BNs.

### 5.1 Synchronism sensitivity in BNs

Following [30], given a BN $f$ of dimension $n$ where, $\forall i \in \{1, \ldots, n\}$, $f_i$ is monotonoic, a positive (resp. negative) edge $(j, i)$ of its influence graph $G(f)$ is *frustrated* in a configuration $x \in \mathbb{B}^n$ iff $x_i \neq x_j$ (resp. $x_i = x_j$). A (directed) cycle in $G(f)$ is *critical* in $x$ iff all its edges are frustrated.

Then, the synchronism sensitivity in BNs can be characterized with respect to their influence graphs as follows.

**Lemma 3 ([30], Prop. 1)** *A critical cycle must be* NOPE*: **n**egative with **o**dd length or **p**ositive with **e**ven length.*

**Theorem 4 ([30])** *Synchronism-sensitivity, i.e., the presence of some synchronous transition that cannot be sequentialized, in a locally monotonic BN $f$ requires the existence of a critical cycle, and thus of a* NOPE*-cycle in its influence graph $G(f)$.*

### 5.2 Synchronism Sensitivity in RPNs

Given any safe RPN $N = (P, T, pre, cont, post, M_0)$, call a pair $(\tau, M) \in 2^T \times 2^P$ such that $\tau$ is s-enabled but not a-enabled in $M$ a *witness of synchronism sensitivity* or, following [30], *normal*.

As in [8], we say for any two transitions $t_1, t_2 \in T$ that $t_1$ *preempts*[1] $t_2$, written $t_1 \rightsquigarrow t_2$ iff the context of $t_2$ intersects the preset of $t_1$:

$$t_1 \rightsquigarrow t_2 \overset{\Delta}{\iff} {}^\bullet t_1 \cap \underline{t_2}.$$

[1] for readers familiar with [8]: we will only need this *immediate* preemption relation $\rightsquigarrow$ here, not the full asymmetric conflict obtained by adding causal precedence

**Theorem 5** *Let $(\tau, M) \in 2^T \times 2^P$ such that $M$ s-enables $\tau$.*

1. *If $\tau = \{t_1, \ldots, t_n\}$ is a* preemption cycle*, i.e.,*

$$t_1 \rightsquigarrow t_2 \rightsquigarrow \ldots t_{n-1} \rightsquigarrow t_n,$$

   *then $(\tau, M)$ is normal.*
2. *Conversely, if $(\tau, M)$ is normal, then $\tau$ contains a preemption cycle.*

*Proof* Part 1 follows immediately from the assumptions. For Part 2, take any transition $t_1 \in \tau$. If there is no place $p \in \underline{t}$ such that $p \in {}^\bullet t_2$ for some $t_2 \in \tau$, remove $t_1$ from $\tau$ and start over. Otherwise, we have $t_2 \rightsquigarrow t_1$, and inspect $({}^\bullet \underline{t_2})$ as above. Since $|\tau| = n$, this process terminates after at most $n$ steps, yielding either a decomposition of $\tau$, or a preemption chain of length at most $n$, or a preemption cycle of length at most $n$. Only the last case corresponds to $\tau$ being normal. □

As an immediate consequence, we note the following minimality result:

**Corollary 2** *Let $\tau$ be such that $(\tau, M)$ is normal, but every $\emptyset \subset \tau' \subseteq \tau$ (with proper inclusions) is a-enabled, i.e., $(\tau', M)$ is not normal. Then $\tau$ is a minimal preemption cycle.*

In Fig. 6, $\tau = \{1{\downarrow}, 2{\downarrow}, 3{\downarrow}\}$ illustrates a preemption cycle, which is also normal in the marking shown; $\tau' = \{1{\uparrow}, 2{\uparrow}, 3{\uparrow}\}$ is another preemption cycle which is not enabled, but would become enabled after firing $\tau$. In Fig. 7, $\tau'' = \{1{\uparrow}, 2{\downarrow}\}$ is a preemption cycle, which is normal in the marking shown.

### 5.3 Application to RPNs encoding BNs

We now study how the characterization of synchronism sensitivity carries over to RPNs which encode BNs following the transformation described in Sect. 4.1. Remember that in this setting, each transition $t$ of the RPN satisify ${}^\bullet t = \{p\}$ and $t^\bullet = \{\overline{p}\}$ with $var(p) = var(\overline{p})$ and $val(p) + val(\overline{p}) = 1$. Thus, $t$ corresponds either to an up-transition $\mathbf{up}(var(p))$ iff $val(p) = 0$ (i.e., $val(\overline{p}) = 1$), or to a down-transition $\mathbf{dw}(var(p))$ iff $val(p) = 1$ (i.e., $val(\overline{p}) = 0$).

Let us assume that the contexts of transitions are minimal, i.e., the DNF being the disjunction of all the context of all the up- (resp. down-) transitions of a node is minimal. Given an up-transition $t = \mathbf{up}(\mathsf{v}_i)$ (resp. a down-transition $t = \mathbf{dw}(\mathsf{v}_i)$) of a node $\mathsf{v}_i$, each place $p \in \underline{t}$ corresponds to a node $\mathsf{v}_j$ with $var(p) = \mathsf{v}_j$. Then, the sign of the influence from $\mathsf{v}_j$ to $\mathsf{v}_i$ is positive if $val(p) = 1$ (resp. $val(p) = 0$) and negative otherwise.

Consider a preemption cycle $t_1 \rightsquigarrow \ldots \rightsquigarrow t_n \rightsquigarrow t_1$, and any arc $(t_i, t_{i+1})$, identifying $i = 1$ and $i = n + 1$ in this cycle. By definition, there exists a place $p \in P$ with $var(p) = \mathsf{v}_j = var(t_i)$ such that $\{p\} = {}^\bullet t_i \cap \underline{t_{i+1}}$, and a place $q \in P$ with $var(q) = \mathsf{v}_k = var(t_{i+1})$ and $\{q\} = {}^\bullet t_{i+1}$. If $t_i = \mathbf{up}(\mathsf{v}_j)$ (i.e., $val(p) = 0$), and $t_{i+1} = \mathbf{dw}(\mathsf{v}_k)$ (i.e, $val(q) = 1$), we say that the type of $(t_i, t_{i+1})$ is $0 - 1$, written $[\nearrow]$, and witnesses a positive influence of $var(t_i)$ on

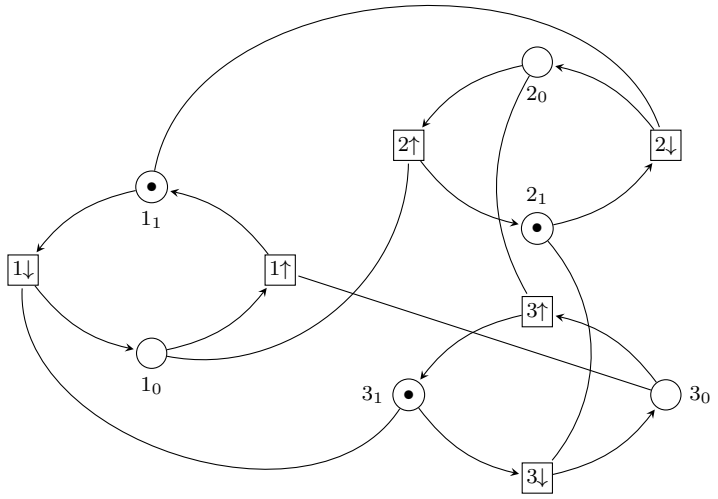


**Fig. 6** A translation of the BN $\langle f_1(x) = \neg x_3, f_2(x) = \neg x_1, \neg f_3(x)\rangle = x_3$ and the configuration 111 into RPN. The step $\tau = \{1\downarrow, 2\downarrow, 3\downarrow\}$ is normal and reflects the negative-odd cycle of the BN.

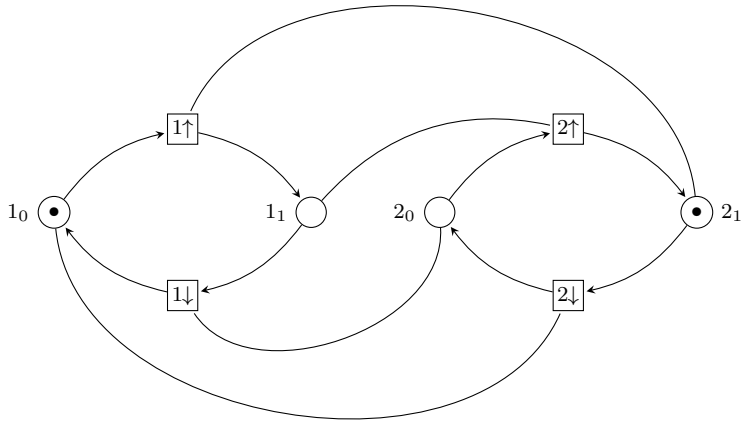


**Fig. 7** A translation of the BN $\langle f_1(x) = x_2, f_2(x) = x_1\rangle$ and configuration 01 into RPN.

$var(t_{i+1})$. Similarly, if $t_i = \mathbf{dw}(\mathsf{v}_j)$ and $t_{i+1} = \mathbf{up}(\mathsf{v}_k)$, the type of $(t_i, t_{i+1})$ is $1-0$, written $[\searrow]$, witnessing a positive influence of $var(t_i)$ on $var(t_{i+1})$; if $t_i = \mathbf{up}(\mathsf{v}_j)$ and $t_{i+1} = \mathbf{up}(\mathsf{v}_k)$, the type of $(t_i, t_{i+1})$ is $0-0$, written $[\multimap]$, witnessing a negative influence of $var(t_i)$ on $var(t_{i+1})$; and if $t_i = \mathbf{dw}(\mathsf{v}_j)$ and $t_{i+1} = \mathbf{dw}(\mathsf{v}_k)$, the type of $(t_i, t_{i+1})$ is $1-1$, written $[\nrightarrow]$, witnessing a negative influence of $var(t_i)$ on $var(t_{i+1})$.

As a consequence, in any preemption cycle, the number of type $[\nearrow]$ arcs and of type $[\searrow]$ arcs must be equal, while nothing can be said in general about the number of $[\multimap]$ and $[\nrightarrow]$ arcs. Since $[\multimap]$ and $[\nrightarrow]$ correspond to arcs with negative signs in the BN's influence graph, adding them in a cycle does not change the cycle's NOPE status (it only changes from negative-odd to positive-even, or vice versa).

**Lemma 4** *Let $\{t_1, \ldots, t_n\}$ be a preemption cycle in $\tau$. Then the product of the signs of associated arcs $(t_i, t_{i+1})$ for $i \in \{1, \ldots, n-1\}$ and $(t_n, t_1)$ is positive iff $n$ is even.*

*Proof* By construction, the types of adjacent arcs have to match: type $[\nearrow]$ and type $[\nrightarrow]$ arcs can only be followed by $[\searrow]$ or $[\nrightarrow]$, and analogously, types $[\multimap]$ and $[\searrow]$ need a successor arc of type $[\nearrow]$ or $[\multimap]$. Hence the word $w \in \{[\multimap], [\nearrow], [\searrow][\nrightarrow]\}^*$ associated to the preemption cycle must not contain the infixes $[\multimap][\searrow]$, $[\nearrow][\nearrow]$, $[\nearrow][\multimap]$, $[\searrow][\searrow]$, $[\searrow][\nrightarrow]$ or $[\nrightarrow][\nearrow]$, and not even $[\multimap][\nrightarrow]$ or $[\nrightarrow][\multimap]$. Since $w$ also has to be cyclic, this implies that

1. between any occurrences of $[\multimap]$ and $[\nrightarrow]$ ($[\nrightarrow]$ and $[\multimap]$), at least one occurrence of $[\nearrow]$ ($[\searrow]$) is required;
2. between any two occurrences of $[\nearrow]$ ($[\searrow]$), at least one occurrence of $[\searrow]$ ($[\nearrow]$) is required;

therefore $|w|_{[\nearrow]} = |w|_{[\searrow]}$, which in turn implies the result. □

*Example 1* The preemption cycle $\tau = \{1\downarrow, 2\downarrow, 3\downarrow\}$ in Fig. 6 is of type $[\nrightarrow][\nrightarrow][\nrightarrow]$, that of $\tau' = \{1\uparrow, 2\uparrow, 3\uparrow\}$ of type $[\multimap][\multimap][\multimap]$; the preemption cycle $\tau'' = \{1\uparrow, 2\downarrow\}$ in Fig. 7 is of type $[\nearrow][\searrow]$.

## 6 Encoding the Interval Semantics with Boolean Networks

In this section, we show how the interval semantics for RPNs (Sect. 3.3.3) can be modelled using BNs with *asynchronous* updating. The resulting BNs subsume the generalized asynchronous updating mode, and enable new reachable configurations, while preserving important dynamical and structural (influence graph) properties.

The interval semantics relies on decomposing the firing of transitions in two stages: a first stage checks the pre-conditions and commits the transition, and a second stage eventually applies the transition (consuming and producing tokens). Because of this decomposition, the interval semantics adds the possibility to trigger transitions which become enabled during the firing of other

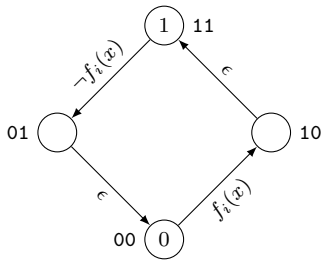


**Fig. 8** Automaton of the value change of a node $i$ in the interval semantics. The states marked 0 and 1 represent the value 0 and 1 of the node. The labels $f_i(x)$ and $\neg f_i(x)$ on edges are the conditions for firing the transitions; $\epsilon$ indicates that the transitions can be done without condition. The states are labeled by the corresponding values of nodes $(2i-1)(2i)$ in our encoding.

transitions. Essentially, its application to BNs can be modelled as follows. Each node $i \in \{1, \ldots, n\}$ is decoupled in two nodes: a "write" node storing the next value $(2i-1)$ and a "read" node for the current value $(2i)$. The decoupling is used to store an ongoing value change, while other nodes of the system still read the current (to be changed) value of the node. A value change is then performed according to the automaton given in Fig. 8: assuming we start in both write and read node with value 0, if $f_i(x)$ is true, then the write node is updated to value 1. The read node is updated in a second step, leading to the value where both write and read nodes are 1. Then, if $f_i(x)$ is false, the write node is updated first, followed, in a second stage by the update of the read node.

Once the write node $(2i-1)$ has changed its value, it can no longer revert back until the read node has been updated. Hence, if $f_i(x)$ becomes false in the intermediate value 10, the read node will still go through value 1 (possibly enabling transitions) before the write node can be updated to 0, if still applicable.

### 6.1 Encoding

From the automaton given in Fig. 8, one can derive Boolean functions for the write $(2i-1)$ and read $(2i)$ nodes. It results in the following BN $\tilde{f}$, encoding the interval semantics for the BN $f$:

**Definition 17 (Interval semantics for Boolean networks)** Given a BN $f$ of dimension $n$, $\tilde{f}$ is a BN of dimension $2n$ where $\forall i \in \{1, \ldots, n\}$,

$$\tilde{f}_{2i-1}(z) \triangleq (f_i(\gamma(z)) \wedge (\neg z_{2i} \vee z_{2i-1})) \vee (\neg z_{2i} \wedge z_{2i-1})$$
$$\tilde{f}_{2i}(z) \triangleq z_{2i-1}$$

where $\gamma(z) \in \mathbb{B}^n$ is defined as $\gamma(z)_i \triangleq z_{2i}$ for every $i \in \{1, \ldots, n\}$.

Given $x \in \mathbb{B}^n$, $\alpha(x) \in \mathbb{B}^{2n}$ is defined as $\alpha(x)_{2i-1} = \alpha(x)_{2i} \stackrel{\Delta}{=} x_i$ for every $i \in \{1, \ldots, n\}$.
A configuration $z \in \mathbb{B}^{2n}$ is called *consistent* when $\alpha(\gamma(z)) = z$.

The function $\gamma : \mathbb{B}^{2n} \to \mathbb{B}^n$ maps a configuration of the interval semantics to a configuration of the BN $f$ by projecting on the read nodes. The function $\alpha : \mathbb{B}^n \to \mathbb{B}^{2n}$ gives the interval semantics configuration of a configuration of the Boolean network $f$, where the read and write nodes have a consistent value.

The correctness of our encoding is given with respect to the interval semantics applied to the RPN translation of the BN. It follows from the correspondence between split transitions of the RPN and update of read and write nodes of the encoded BN: for any Petri net transition $t$ of the RPN $(\!|f|\!)$, the triggering of $t^-$ matches with the update of the "write node" for $var(t^\bullet)$ of the BN, and the triggering of $t^+$ matches with the update of the "read node" for $var(t^\bullet)$ of the BN.

**Theorem 6** *Given a BN $f$ of dimension $n$, for all $x, y \in \mathbb{B}^n$,*

$$(\!|x|\!) \xrightarrow[\text{istep}]{(\!|f|\!)}{}^* (\!|y|\!) \iff \alpha(x) \xrightarrow[\text{async}]{\tilde{f}}{}^* \alpha(y) \ .$$

## 6.2 Consistency

The above theorem shows that the asynchronous updating of the BN $\tilde{f}$ encoding the interval semantics can reproduce any behaviour of the generalized asynchronous updating of $f$. The aim of this section is to show that the interval semantics still preserves important constraints of the BN on its dynamics. In particular, we show the one-to-one relationship between the fixpoints of the BN and its encoding for interval semantics; and that the influences are preserved with their sign.

Lemma 5 states that from any configuration of encoded BN, one can always reach a consistent configuration:

**Lemma 5 (Reachability of consistent configurations)** *For any $z \in \mathbb{B}^{2n}$ such that $\alpha(\gamma(z)) \neq z$, $\exists y \in \mathbb{B}^n : z \xrightarrow[\text{async}]{\tilde{f}}{}^* \alpha(y)$.*

*Proof* For each $i \in \{1, \ldots, n\}$ such that $z_{2i-1} \neq z_{2i}$, we update the $2i$ node, in arbitrary order. This leads to the configuration $z' \in \mathbb{B}^{2n}$ where $\forall i \in \{1, \ldots, n\}$, $z'_{2i} = z'_{2i-1} = z_{2i-1}$. Hence, by picking $y = \gamma(z)$, we obtain $z \xrightarrow[\text{async}]{\tilde{f}}{}^* \alpha(y)$. □

The one-to-one relationship between fixpoints of $f$ and fixpoints of $\tilde{f}$ is given by the following lemma:

**Lemma 6 (Fixpoint equivalence)** *$\forall x \in \mathbb{B}^n$, $f(x) = x \Rightarrow f(\alpha(x)) = \alpha(x)$; and $\forall z \in \mathbb{B}^{2n}$, $\tilde{f}(z) = z \Rightarrow \alpha(\gamma(z)) = z \wedge f(\gamma(z)) = \gamma(z)$.*

*Proof* Let $x \in \mathbb{B}^n$ be such that $f(x) = x$. We have that $\alpha(x)_{2i-1} = \alpha(x)_{2i} = x_i = f_i(x)$. Hence, $\tilde{f}_{2i-1}(\alpha(x)) = f_i(\gamma(\alpha(x))) = f_i(x) = \alpha(x)_{2i-1}$; and $\tilde{f}_{2i}(\alpha(x)) = \alpha(x)_{2i-1} = \alpha(x)_{2i}$. Thus, $\tilde{f}(\alpha(x)) = \alpha(x)$.

Let $z \in \mathbb{B}^{2n}$ be such that $\tilde{f}(z) = z$. For each $i \in \{1, \ldots, n\}$, because $\tilde{f}_{2i}(z) = z_{2i}$, by the definition of $\tilde{f}_{2i}$, we obtain that $z_{2i} = z_{2i-1}$. Thus, $\alpha(\gamma(z)) = z$. Moreover, as $(\neg z_{2i} \vee z_{2i-1})$ reduces to true and $(\neg z_{2i} \wedge z_{2i-1})$ reduces to false, $\tilde{f}_{2i-1}(z) = f_i(\gamma(z)) = z_{2i-1} = \gamma(z)_i$. Therefore, $f(\gamma(z)) = \gamma(z)$. □

### 6.3 Influence graph

As defined in Sect. 2, the influence graph provides a summary of the causal dependencies between the value changes of nodes of the BN. We show that our encoding of interval semantics preserves the causal dependencies of the original network, and in particular, preserves the cycles and their signs.

From the definition of $\tilde{f}$, one can derive that all the influences in $f$ are preserved in $\tilde{f}$, and no additional influences between different variables $i, j$ are created by the encoding. This latter fact is addressed by the following lemma:

**Lemma 7** *For any $i, j \in \{1, \ldots, n\}$, $i \neq j$, there is a positive (resp. negative) edge from $j$ to $i$ in $G(f)$ if and only if there is a positive (resp. negative) edge from $2j$ to $2i-1$ in $G(\tilde{f})$.*

*Proof* Let us define $x, y \in \mathbb{B}^n$ such that $\Delta(x, y) = \{j\}$, and $z, z' \in \mathbb{B}^{2n}$ such that $z = \alpha(x)$ and $\Delta(z, z') = \{2j\}$, i.e., $z'_{2j} = y_j$. Because $z_{2i} = z_{2i-1}$ and, as $i \neq j$, $z'_{2i} = z'_{2i-1}$, we obtain that $\tilde{f}_{2i-1}(z) = f_i(x)$ and $\tilde{f}_{2i-1}(z') = f_i(y)$. □

**Lemma 8** *For any $i \in \{1, \ldots, n\}$,*

*a. there is a positive self-loop on $2i-1$ in $G(\tilde{f})$ if and only if there exists $x \in \mathbb{B}^n$ such that $f_i(x) = x_i$;*
*b. there is never a negative self-loop on $2i-1$ in $G(\tilde{f})$;*
*c. there is never a positive edge from $2i$ to $2i-1$ in $G(\tilde{f})$;*
*d. there is a negative edge from $2i$ to $2i-1$ in $G(\tilde{f})$ if and only if there exists $x \in \mathbb{B}^n$ such that $f_i(x) \neq x_i$*
*e. there is always exactly one edge from $2i-1$ to $2i$ in $G(\tilde{f})$ and it is positive.*

*Proof* (a) Let us consider $z, z' \in \mathbb{B}^{2n}$ such that $\Delta(z, z') = \{2i-1\}$ with $z_{2i-1} = 0$: $\tilde{f}_{2i-1}(z) = 0 = \neg\tilde{f}_{2i-1}(z') \Leftrightarrow [(z_{2i} = 0 \wedge f_i(\gamma(z)) = 0) \vee (z_{2i} = 1 \wedge f_i(\gamma(z)) = 1)] \Leftrightarrow f_i(\gamma(z)) = z_{2i}$. (b) Let us consider $z, z' \in \mathbb{B}^{2n}$ such that $\Delta(z, z') = \{2i-1\}$ with $z_{2i-1} = 0$ and $\tilde{f}_{2i-1}(z) = 1 = \neg\tilde{f}_{2i-1}(z')$. Thus, $z_{2i} = 0$, therefore, $\tilde{f}_{2i-1}(z') = z'_{2i-1} = 1$, which is a contradiction. (c) Let us consider $z, z' \in \mathbb{B}^{2n}$ such that $\Delta(z, z') = \{2i\}$ with $z_{2i} = 0$: if $z_{2i-1} = z'_{2i-1} = 0$, then $\tilde{f}_{2i-1}(z) \geq \tilde{f}_{2i-1}(z')$; if $z_{2i-1} = z'_{2i-1} = 1$, then $\tilde{f}_{2i-1}(z) \geq \tilde{f}_{2i-1}(z')$; therefore there cannot be a negative edge from $2i$ to $2i-1$ in $G(\tilde{f})$. (d) $\exists z, z' \in \mathbb{B}^{2n}$: $\Delta(z, z') = \{2i\}$, $z_{2i} = 0$, $\tilde{f}_{2i-1}(z) = 1 = \neg\tilde{f}_{2i-1}(z') \Leftrightarrow [(z_{2i-1} = z'_{2i-1} = 0 \wedge f_i(\gamma(z)) = 1) \vee (z_{2i-1} = z'_{2i-1} = 1 \wedge f_i(\gamma(z')) = 0)] \Leftrightarrow \exists x \in \mathbb{B}^n : f_i(x) = \neg x_i$. (e) By $\tilde{f}_{2i}$ definition. □

From Lemma 8, one can deduce that if there is a positive self-loop on $i$ in $G(f)$, then there is a positive self-loop on $2i-1$ in $G(\tilde{f})$; and if there is a negative self-loop on $i$ in $G(f)$, then there is a negative edge from $2i$ to $2i-1$ in $G(\tilde{f})$.

We can then deduce that the positive and negative cycles of $G(f)$ are preserved in $G(\tilde{f})$. It is worth noting that the encoding may also introduce negative cycles between $2i-1$ and $2i$ and positive self-loops on $2i-1$, for some $i \in \{1, \ldots, n\}$.

**Lemma 9** *To each positive (resp. negative) cycle in $G(f)$ of length $k > 1$, there exists a corresponding positive (resp. negative) cycle in $G(\tilde{f})$ of length $2k$. To each positive self-loop in $G(f)$ corresponds one positive self-loop in $G(\tilde{f})$; to each negative self-loop in $G(f)$ corresponds a negative cycle in $G(\tilde{f})$ of length 2.*

*Proof* For cycle of length $k > 1$, by Lemma 7 and by the fact that there is a positive edge from $2i-1$ to $2i$ in $G(\tilde{f})$: each edge $(i, j)$ in the cycle in $G(f)$ is mapped to the string $(2i, 2j-1)(2j-1, 2j)$, giving a cycle in $G(\tilde{f})$ of the same sign. Correspondence of self-loops is given by Lemma 8. □

## 7 Beyond Generalized Asynchronicity and Interval Semantics

BNs are widely used to model the qualitative dynamics of biological networks, notably of signalling and gene regulation networks.

A major concern is the impact of the chosen updating mode on the validation of the model. Indeed, it is usual to assess the accordance of a BN with measurement data, including time series: it is expected that the observed behaviours can be reproduced in the abstract model. With this perspective, the computation of reachable configurations in BNs is key. For example, let us assume we observe (in the concrete system) that a given component (e.g., gene) gets eventually activated: if the reachability analysis of the BN concludes that no reachable state has this component active, the model would likely be rejected by the modeller.

In biological applications, the analysis of BNs merely splits into two scientific sub-communities: the one preferring the synchronous updating mode, and the one preferring the asynchronous updating mode. The generalized asynchronous updating, which subsumes synchronous and asynchronous, seems a good compromise but it received very little attention in practice. It should be noted that most of computational tools rely only on either synchronous or asynchronous modes, which can provide a partial explanation.

Is the generalized asynchronous mode the ultimate updating mode when analysing reachable configurations in BNs for biological systems? If little is known on time and speed features of the system and the reachability analysis with generalized asynchronicity concludes on the absence of the observed state, can we safely invalidate the model?

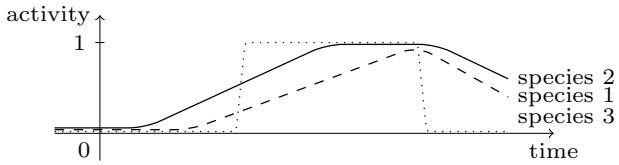


**Fig. 9** A possible evolution of the activity of species modelled by the BN of Fig. 2 (species 1 in dashed line, species 2 plain, species 3 dotted).

In the following motivating example (Sect. 7.1), we show that the generalized asynchronous updating can miss transitions, hence reachable configurations, which correspond to particular, but plausible, behaviours. Thus, the resulting analysis can be misleading on the absence of some behaviours, notably regarding the reachability of attractors (configurations reachable in the long-run), and may lead to rejection of valid models. It is worth noting that the network considered in the example is embedded in many actual models of biological networks, e.g., [28, 29, 43].

As introduced in Sect. 3, the interval semantics of RPNs takes advantage of the fine-grained specification of causality of transitions to enable new behaviours, i.e., new reachable states, which can be caused by specific ordering and duration of updates. We show in Sect. 7.2 that using the encoding of BNs into RPNs provided in Sect. 4.1 and applying the interval semantics correctly recovers the missing reachable configurations in our motivating example.

Finally, in Sect. 7.3 we explore further extensions of the interval semantics resulting in correct over-approximation of the configurations reachable by any multi-valued refinement of the BN.

### 7.1 Motivating example

Let us consider the BN defined in Fig. 2. The BN and its influence graph suggest that the activity of species 3 increases when 1 is inactive and 2 is active. In any scenario starting from 000 where 3 eventually increases, 2 has to increase to trigger the increase of 3. Hence, according to the generalized asynchronous updating represented in Fig. 2 (c), the only transition which represents an increase of 3 is $010 \to 011$. After this, no transition is possible.

But, assuming the BN abstracts continuous evolution of activities, the following scenario, pictured in Fig. 9, becomes possible: initially, the inactivity of species 1 causes an increase of the activity of species 2, represented in plain line on the figure. Symmetrically, the absence of species 2 causes an increase of the activities of species 1 (dashed line). This corresponds to the evolution described by the arrow $000 \to 110$ in Fig. 2(b) and leads to a (transient) configuration where species 1 and 2 are present.

Assume that 1 and 2 activity increase slowly. After some time, however, the activity of 2 becomes sufficient for influencing positively the activity of 3, while there is still too little of species 1 for influencing negatively the activity

of 3. Species 3 can then increase. In the scenario represented in the figure, 3 (dotted line) increases quickly, and then 1 and 2 continue to increase. In summary, the activity of species 3 increased from 0 to 1 *during* the increase of 1 and 2, which was not predicted by the generalized asynchronous updating (Fig. 2(b)).

One could argue that in this case, one should better consider more fine-grained models, for instance by allowing more than binary values on nodes in order to reflect the different activation thresholds. However, the definition of the refined models would require additional parameters (the different activation thresholds) which are unknown in general. Our goal is to allow capturing these behaviours already in the Boolean abstraction, so that any refinement would remove possible transitions, and not create new ones.

### 7.2 Application of the Interval Semantics of RPNs

Let us consider the BN $f$ in Fig. 2 and its RPN encoding $(\!|f|\!)$ in Fig. 5. Starting from the marking $(\!|000|\!)$, $1\uparrow^- \; 2\uparrow^- \; 2\uparrow^+ \; 3\uparrow^- \; 3\uparrow^+ \; 1\uparrow^+$ is a *complete i-run* (Def. 14) of the interval semantics, and leads to the marking $(\!|111|\!)$.

Similarly, let us consider the encoding of the interval semantics in the BN $\tilde{f}$, as defined in Sect. 6. We obtain the following possible sequence of fully asynchronous updates of $\tilde{f}$:

$$00\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,10\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,11\,00$$
$$\xrightarrow[\text{async}]{\tilde{f}} 10\,11\,10 \xrightarrow[\text{async}]{\tilde{f}} 10\,11\,11 \xrightarrow[\text{async}]{\tilde{f}} 11\,11\,11$$

Therefore, with the interval semantics, the configuration 111 of $f$ is reachable from 000, contrary to the generalized asynchronous semantics. This is due to the decoupling of the update of node 1: the activation of 1 is delayed which allows activating node 3 beforehand.

### 7.3 Beyond the Interval Semantics

With the interval semantics, during the interleaving of transitions, the nodes have access only to the before-update value of other nodes. Moreover, the interval semantics enforces the update application: once an update is triggered (write node gets a different value than the read node), no further update on the same node is possible until the update has been applied. Thus, if for instance the update triggers a change of value from 0 to 1, the interval semantics guarantees that the read node will eventually have the value 1.

In terms of modeling, the restriction to before-update values in our interval semantics can be seen as an asymmetry in the consideration of transitions: the resource modified by the transition is still available during the interval of update, whereas the result is only available once the transition finished. When

modelling biological systems, it translates into considering only species which are slow to reach their activity threshold.

Actually, the choice of whether the before-update, after-update or both values are available during the update may be done according to the knowledge of the modeled system. Our construction in Sect. 6 can easily be adapted for giving access, depending on the node, to the after-update value instead of the before-update value. For instance, if the node $j$ should follow closely value changes of node $i$, then node $j$ should access the after-update value (write node) of $i$, whereas, as in our motivating example, if $i$ is slow to update compared to $j$, node $j$ should access the before-update value (read node) of $i$.

### *7.3.1 Most Permissive Fully Asynchronous Semantics for Boolean Networks*

Finally, we consider here a more permissive symmetric version which would allow the access of both before-update and after-update values and do not enforce update application. This choice may be very reasonable when not much is known about the system, for instance about the relative speed of the nodes.

This leads us to define a *most permissive fully asynchronous semantics* for BNs which is defined as a 3-valued semantics in order to represent non instantaneous updates: a component in a configuration can now have value $\frac{1}{2}$, in addition to the usual 0 and 1, and the updates are done in two stages: if the network is in a configuration $x$ where for some $i$, $f_i(x) \neq x_i$, the update of $x_i$ will be in two stages, going through an intermediate configuration $y$ with $y_i = \frac{1}{2}$. In this intermediate configuration $y$, other updates can occur before the completion of the update of node $i$, and they will be allowed to use either the value 0 or 1 for node $i$. In the end, for a 3-valued configuration $x \in \{0, \frac{1}{2}, 1\}^n$, we allow all the intermediate values to be approximated either as 0 or as 1. The possible approximations are defined as the set $Approx(x)$ of Boolean configurations $x' \in \mathbb{B}^n$ such that, for every $i \in \{1, \ldots, n\}$,

- $x'_i = 0$ if $x_i = 0$,
- $x'_i = 1$ if $x_i = 1$,
- otherwise $x'_i$ can be either 0 or 1.

**Definition 18 (Most permissive fully asynchronous semantics for Boolean networks)** Given a BN $f$, the binary irreflexive relation $\xrightarrow[\text{mpa}]{f} \subseteq \{0, \frac{1}{2}, 1\}^n \times \{0, \frac{1}{2}, 1\}^n$ is defined as:

$$x \xrightarrow[\text{mpa}]{f} y \overset{\Delta}{\Longleftrightarrow} \exists i \in \{1, \ldots, n\}, x' \in Approx(x) : \Delta(x, y) = \{i\}$$
$$\wedge\, y_i = \begin{cases} f_i(x') & \text{if } x_i = \frac{1}{2} \\ \frac{1}{2} & \text{otherwise } (x_i \neq f_i(x')) \end{cases} .$$

We write $\xrightarrow[\text{mpa}]{f}{}^*$ for the transitive closure of $\xrightarrow[\text{mpa}]{f}$.

Similarly to the BN encoding of interval semantics presented in Sect. 6, the most permissive fully asynchronous semantics of a BN $f$ of dimension $n$ can be encoded as an asynchronous BN $\tilde{\tilde{f}}$ of dimension $3n$ where each node $i \in \{1, \ldots, n\}$ is decoupled into an after-update value node $(2i-1)$ and a before-update value node $(2i)$. As in Def. 17, the updating of this latter node consists in copying the after-update value node: $\tilde{\tilde{f}}_{2i}(z) \stackrel{\Delta}{=} z_{2i-1}$. The definition of $\tilde{\tilde{f}}_{2i-1}$ is a bit more complex as one has to rewrite $f_i(x)$ to use (non-deterministically) either the before-update or after-update value of input nodes. This non-deterministic choice can be encoded using extra "coin flip" nodes $(2n+j)$ for $j \in \{1, \ldots, n\}$ with $\tilde{\tilde{f}}_{2n+j}(z) \stackrel{\Delta}{=} \neg z_{2n+j}$. Then, assuming $f_i(x)$ is specified using propositional logic, the literals $x_j$ appearing in $f_i(x)$ are replaced with $\tilde{\tilde{x}}_j \stackrel{\Delta}{=} (z_{2n+j} \vee z_{2j}) \wedge (\neg z_{2n+j} \vee z_{2j-1})$. Also, contrary to the interval semantics, the most permissive fully asynchronous semantics does not enforce the update application. Thus, $\tilde{\tilde{f}}_{2i-1}(z) \stackrel{\Delta}{=} [f_i(x)]_{[\tilde{\tilde{x}}_j/x_j, j\in\{1,\ldots,n\}]}(z)$.

### *7.3.2 Most permissive fully asynchronous semantics simulates any multivalued refinement*

Multivalued networks are generalization of BNs where the nodes $x_i$ can take values other than $\{0,1\}$. Let us denote the possible values as $\mathbb{M} \stackrel{\Delta}{=} \{0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1\}$ for some integer $m$. For simplicity in notations, we assume the same number of values for all the nodes. A *configuration* is now a vector $x \in \mathbb{M}^n$. Given two configurations $x, y \in \mathbb{M}^n$, the components that differ are noted $\Delta(x,y) \stackrel{\Delta}{=} \{i \in \{1, \ldots, n\} \mid x_i \neq y_i\}$.

In practical modelling applications, multivalued networks enable considering different thresholds for the interactions from one component to its regulators: for instance, the activation of a second component may require the first component to be only slightly active $(\frac{1}{m})$, whereas the activation of a third component may require the full activation (1) of this first one.

Hence, multivalued networks can be considered as refinements of BNs, where in addition to the logic of interactions, one can mix different thresholds to consider a component active or inactive. This fined-grained specification requires more information on the system, and it is then natural to aim at performing analyses at a more abstract level (BN) and then transfering the results to possible multivalued concretisations of the model.

In this section, we show that the most permissive fully asynchronous semantics enables such a reasoning for reachability properties: essentially, this semantics captures any behaviour possible in any multivalued refinement of the BN *with asynchronous updating*. Therefore, if a configuration is not reachable in the most permissive fully asynchronous semantics, there exists no multivalued refinement for which the configuration become reachable with asynchronous updating.

We illustrate this result with Examples 2 and 3 at the end of the section. Notably, the last one shows an example where both generalized asynchronous

updating and interval semantics of a BN fail to capture behaviours which are actually possible in a multivalued refinement of it; these behaviours are correctly preserved by the most permissive fully asynchronous semantics.

From a specification point of view, multivalued networks can be defined similarly to BNs, except that the functions now map the configurations to either "↑" (increase the value of component by $\frac{1}{m}$), "−" (do not change the value of component), or "↓" (decrease the value of component by $\frac{1}{m}$).

Def. 21 formalizes the notion of multivalued refinement: a multivalued network $F$ refines a BN $f$ if, for every component $i \in \{1, \ldots, n\}$, for each multivalued configuration $x$, if $F_i(x)$ leads to an increase (resp. decrease) of the value of $i$, there is a binarization $x' \in \mathbb{B}^n$ of $x$ such that $f_i(x') = 1$ (resp. $f_i(x') = 0$). Here, the binarization allows to map non-binary values to either 0 or 1.

Theorem 7 states that, given a BN $f$, any *fully asynchronous* transition of any multivalued refinement $F$ of $f$ is captured by the most permissive fully asynchronous semantics, possibly by the mean of several intermediate transitions.

**Definition 19 (Multivalued network)** A *multivalued network* of dimension $n$ over a value range $\mathbb{M} = \{0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1\}$ is a collection of functions $F = \langle F_1, \ldots, F_n \rangle$ where $\forall i \in \{1, \ldots, n\}, F_i : \mathbb{M}^n \to \{\uparrow, -, \downarrow\}$.

**Definition 20 (Asynchronous updating in multivalued networks)** Given a multivalued network $F$, the binary irreflexive relation $\xrightarrow[\text{async}]{F} \subseteq \mathbb{M}^n \times \mathbb{M}^n$ is defined as:

$$x \xrightarrow[\text{async}]{F} y \overset{\Delta}{\Longleftrightarrow} \exists i \in \{1, \ldots, n\} : \Delta(x, y) = \{i\}$$
$$\wedge\, y_i = \begin{cases} \min\{0, x_i - \frac{1}{m}\} & \text{if } F_i(x) = \downarrow \\ \max\{1, x_i + \frac{1}{m}\} & \text{if } F_i(x) = \uparrow \end{cases}.$$

We write $\xrightarrow[\text{async}]{F}{}^*$ for the transitive closure of $\xrightarrow[\text{async}]{F}$.

We now define a notion of *multivalued refinement* of a BN, which formalizes the intuition that the moves defined by the multivalued network are compatible with those of the BN.

**Definition 21 (Multivalued refinement)** A multivalued network $F$ of dimension $n$ over a value range $\mathbb{M} = \{0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1\}$ *refines* a BN $f$ of equal dimension $n$ iff for every configuration $x \in \mathbb{M}^n$ and every $i \in \{1, \ldots, n\}$:

- $F_i(x) = \uparrow \implies \exists x' \in \mathit{Approx}(x) : f_i(x') = 1$
- $F_i(x) = \downarrow \implies \exists x' \in \mathit{Approx}(x) : f_i(x') = 0$

where $\mathit{Approx}$ is generalized to multi-valued networks by $\mathit{Approx}(x) \overset{\Delta}{=} \mathit{Approx}(\mathit{abstr}(x))$ with $\mathit{abstr} : \mathbb{M}^n \to \{0, \frac{1}{2}, 1\}^n$ mapping every configuration of the multivalued network into a 3-valued configuration, which is defined for every $i \in \{1, \ldots, n\}$ as:

- $abstr(x)_i \stackrel{\Delta}{=} 0$ if $x_i = 0$,
- $abstr(x)_i \stackrel{\Delta}{=} 1$ if $x_i = 1$,
- $abstr(x)_i \stackrel{\Delta}{=} \frac{1}{2}$ otherwise.

**Theorem 7 (Most permissive fully asynchronous semantics simulates any multivalued refinement)** *Let $f$ be a BN of dimension $n$ and $F$ a multivalued refinement of $f$. Then*

$$\forall x, y \in \mathbb{M}^n, \quad x \xrightarrow[\text{async}]{F} y \implies abstr(x) \xrightarrow[\text{mpa}]{f}{}^{*} abstr(y)\,.$$

*Proof* We assume first that $m > 1$. By definition of $\xrightarrow[\text{async}]{F}$ for multivalued networks, there exists a unique $i$ such that $\Delta(x, y) = \{i\}$. Then we have to study the different cases determined by the value of $x_i$ and of $F_i(x)$.

The first case is $0 < x_i < \frac{m-1}{m}$ and $F_i(x) = \uparrow$. It implies $y_i = x_i + \frac{1}{m}$, and we observe that, in this case, $abstr(x) = abstr(y)$. Then trivially $abstr(x) \xrightarrow[\text{mpa}]{f}{}^{*} abstr(y)$. The case of $\frac{1}{m} < x_i < 1$ and $F_i(x) = \downarrow$ is symmetric.

The other cases are all similar; consider for instance $x_i = 0$ and $F_i(x) = \uparrow$, which imposes $y_i = \frac{1}{m}$. Notice first that $\Delta(abstr(x), abstr(y)) = \{i\}$ and $abstr(x)_i = 0$ and $abstr(y)_i = \frac{1}{2}$. Now, since $F$ is a multivalued refinement of $f$, then by Def. 21, there exists an $x' \in Approx(x) = Approx(abstr(x))$ such that $f_i(x') = 1$. Thus, we get $abstr(x) \xrightarrow[\text{mpa}]{f} abstr(y)$. The case when $x_i = 1$ and $F_i(x) = \downarrow$ is similar. Regarding the case when $y_i = 1$ and $F_i(x) = \uparrow$, note that $abstr(x)_i = \frac{1}{2}$ and $abstr(y)_i = 1$ and that there exists an $x' \in Approx(x) = Approx(abstr(x))$ such that $f_i(x') = 1$. Thus, $abstr(x) \xrightarrow[\text{mpa}]{f} abstr(y)$. The case when $y_i = 0$ and $F_i(y) = \downarrow$ is similar.

Finally, for $m = 1$, consider the case where $x_i = 0$ and $F_i(x) = \uparrow$, which imposes $y_i = 1$. Now $abstr(x) = x$ and $abstr(y) = y$. In the most permissive fully asynchronous semantics, we have $x \xrightarrow[\text{mpa}]{f} z \xrightarrow[\text{mpa}]{f} y$ with an intermediate state $z$ defined by $\Delta(x, z) = \{i\}$ and $z_i = \frac{1}{2}$. The transition $z \xrightarrow[\text{mpa}]{f} y$ is allowed because $x \in Approx(z)$. □

*Example 2* The scenario pictured in Fig. 9 can be obtained as a behaviour of a 3-level refinement $F$ of the BN $f$ in Fig. 2, with the following update functions:

$$F_1(x) \stackrel{\Delta}{=} \uparrow \text{ if } x_2 < 1 \text{ else } \downarrow$$
$$F_2(x) \stackrel{\Delta}{=} \uparrow \text{ if } x_1 < 1 \text{ else } \downarrow$$
$$F_3(x) \stackrel{\Delta}{=} \uparrow \text{ if } x_1 \leq \frac{1}{2} \wedge x_2 \geq \frac{1}{2} \text{ else } \downarrow$$

We get $000 \xrightarrow[\text{async}]{F} 0\frac{1}{2}0 \xrightarrow[\text{async}]{F} \frac{1}{2}\frac{1}{2}0 \xrightarrow[\text{async}]{F} \frac{1}{2}\frac{1}{2}\frac{1}{2} \xrightarrow[\text{async}]{F} \frac{1}{2}\frac{1}{2}1 \ldots$.

In particular, imagine that a fourth species would activate when $x_1$, $x_2$ and $x_3$ are all $\geq \frac{1}{2}$, then even the generalized asynchronous updating mode would not capture its activation, contrary to our interval semantics for BNs.

*Example 3* Let us consider the BN $f$ of dimension 3 defined as follows:

$$\begin{aligned} f_1(x) &\stackrel{\Delta}{=} 1 \\ f_2(x) &\stackrel{\Delta}{=} x_1 \\ f_3(x) &\stackrel{\Delta}{=} x_2 \wedge \neg x_1 \end{aligned}$$

Starting from configuration 000 the generalized asynchronous mode allows only the following transitions $000 \xrightarrow[\text{gen}]{f} 100 \xrightarrow[\text{gen}]{f} 110$, where 110 is a fixpoint of $f$. The interval semantics lead to a very similar behaviour, with the following unique sequence of asynchronous transitions of the BN encoding of the interval semantics:

$$00\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 11\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 11\,10\,00 \xrightarrow[\text{async}]{\tilde{f}} 11\,11\,00$$

Indeed, in order to activate species 2, 1 has to be activated first as in the interval semantics species 2 only has access to the before-update value of 1. Then, once species 1 is active, it is impossible to activate species 3.

Now, let us consider the following 3-level refinement $F$ of the BN $f$:

$$\begin{aligned} F_1(x) &\stackrel{\Delta}{=} \uparrow \\ F_2(x) &\stackrel{\Delta}{=} \uparrow \text{ if } x_1 \geq \frac{1}{2} \text{ else } \downarrow \\ F_3(x) &\stackrel{\Delta}{=} \uparrow \text{ if } x_2 \geq \frac{1}{2} \wedge x_1 \leq \frac{1}{2} \text{ else } \downarrow \end{aligned}$$

The following asynchronous transitions are possible from configuration 000: $000 \xrightarrow[\text{async}]{F} \frac{1}{2}00 \xrightarrow[\text{async}]{F} \frac{1}{2}\frac{1}{2}0 \xrightarrow[\text{async}]{F} \frac{1}{2}\frac{1}{2}\frac{1}{2}$. These transitions are also transitions of the most permissive fully asynchronous semantics of $f$, $\xrightarrow[\text{mpa}]{f}$. Essentially, as in this semantics species can have access to either the before-update or after-update value of other species, species 2 can be activated by reading the after-update value of 1, while species 3 can be activated by reading the before-update value of 1. An example of possible sequence of asynchronous transitions of the BN encoding of the most permissive fully asynchronous semantics is the following:

$$\begin{aligned} 00\,00\,00 \xrightarrow[\text{async}]{\tilde{\tilde{f}}} 10\,00\,00 \xrightarrow[\text{async}]{\tilde{\tilde{f}}} 10\,10\,00 \xrightarrow[\text{async}]{\tilde{\tilde{f}}} 10\,11\,00 \\ \xrightarrow[\text{async}]{\tilde{\tilde{f}}} 10\,11\,10 \xrightarrow[\text{async}]{\tilde{\tilde{f}}} 10\,11\,11 \end{aligned}$$

As in the previous example, let us consider a fourth species activated when $x_1$, $x_2$, and $x_3$ are all greater or equal than $\frac{1}{2}$: such an activation is captured neither by the generalized asynchronous updating nor by the interval semantics of the abstract BN $f$, whereas it is captured by its most permissive fully asynchronous semantics.

## 8 Discussion

With this paper, we detailed the link between Boolean Networks (BNs) and Read (or contextual) Petri Nets (RPNs) by focusing on the analysis of concurrency enabled by the latter framework. On the one hand, BNs have prominent structural properties between the components and their evolution, while on the other hand RPNs bring a fine-grained specification of the causality and effect of transitions. We show how we can take benefit of both approaches to first bring new updating modes to BNs by encoding RPN semantics, and, secondly, propose further extensions of these semantics aiming at obtaining correct Boolean abstractions of discrete dynamical systems.

To sum up, the contributions of this paper include:

- The encoding of BNs into RPNs, similar to other encodings already existing in the literature, here specialized for Read Petri nets;
- The encoding of RPNs into BNs, which allows a brief proof by reduction of the PSPACE-completeness of the reachability decision in asynchronous BNs;
- A generic characterization of synchronism sensitivity in RPNs, which when instantiated to BN translations, allows to recover a recent result in BNs;
- The encoding of the interval semantics of RPNs as asynchronous BNs, enabling new behaviours missed by usual BN updating modes;
- An extension of the interval semantics for BNs which guarantees to include the behaviour of any multivalued refinement.

For practical applications, the thorough link between BNs and Petri nets enables the use of conceptual tools based on causality and concurrency, such as unfoldings offering more compact representation of behaviours [20,7,16,27] and for which efficient software tools have been developed for safe PNs [38] and RPNs [36]. For example, [13,16] show the applicability of unfoldings to analyse reachable states and attractors in BNs with biological use cases having up to 88 components.

The transitions enabled by the interval and most permissive semantics are due to nodes which update at different time scales. For instance with the interval semantics, whenever committed to a value change, in the meantime of the update application, the other nodes of the network still evolve subject to its before-update value. This time scale consideration brings an interesting feature when modeling biological networks which gathers processes of different nature and velocity. Our encodings can be applied only to a subset of nodes, offering

a flexible modelling approach. Moreover, because the encodings rely on asynchronous BNs, they can be implemented using any software tools supporting the asynchronous updating mode.

The introduction of the most permissive fully asynchronous semantics for BNs motivates future work to determine if it offers the smallest abstraction of any multivalued refinement (i.e., to any transition of the most permissive semantics corresponds an asynchronous transition of a multivalued refinement), and to assess the complexity of reachability decision. Finally, further work may explore links between BNs and RPNs with real-time semantics [6], aiming at tightening connections between the two hybrid frameworks.

## Acknowledgements

The authors acknowledge the support from the French Agence Nationale pour la Recherche (ANR), in the context of the ANR-FNR project "AlgoReCell" ANR-16-CE12-0034, from the Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02), from Paris Ile-de-France Region (DIM RFSI), and from UMI 2000 ReLaX (CNRS, Univ. Bordeaux, ENS Paris-Saclay, CMI, IMSc) for the internship of Aalok Thakkar at ENS Paris-Saclay, at that time student at Chennai Mathematical Institute, India, and during which part of this work was done.

## References

1. Aracena, J.: Maximum number of fixed points in regulatory boolean networks. Bulletin of Mathematical Biology **70**(5), 1398–1409 (2008). DOI 10.1007/s11538-008-9304-7
2. Aracena, J., Demongeot, J., Goles, E.: Positive and negative circuits in discrete neural networks. IEEE Transactions of Neural Networks **15**, 77–83 (2004)
3. Aracena, J., Goles, E., Moreira, A., Salinas, L.: On the robustness of update schedules in Boolean networks. Biosystems **97**(1), 1 – 8 (2009). DOI 10.1016/j.biosystems.2009.03.006
4. Aracena, J., Richard, A., Salinas, L.: Number of fixed points and disjoint cycles in monotone boolean networks. SIAM Journal on Discrete Mathematics **31**(3), 1702–1725 (2017)
5. Baetens, J., der Weeën, P.V., Baets, B.D.: Effect of asynchronous updating on the stability of cellular automata. Chaos, Solitons & Fractals **45**(4), 383 – 394 (2012). DOI 10.1016/j.chaos.2012.01.002
6. Balaguer, S., Chatain, T., Haar, S.: A concurrency-preserving translation from time Petri nets to networks of timed automata. Formal Methods in System Design **40**(3), 330–355 (2012). DOI 10.1007/s10703-012-0146-4
7. Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., Schwoon, S.: Efficient unfolding of contextual Petri nets. TCS **449**, 2–22 (2012)
8. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri nets, asymmetric event structures, and processes. Information and Computation **171**(1), 1–49 (2001)
9. Busi, N., Pinna, G.M.: Non sequential semantics for contextual P/T nets. In: Application and Theory of Petri Nets, *Lecture Notes in Computer Science*, vol. 1091, pp. 113–132. Springer (1996)
10. Chaouiya, C.: Petri net modelling of biological networks. Briefings in Bioinformatics **8**(4), 210–219 (2007). DOI 10.1093/bib/bbm029

11. Chaouiya, C., Naldi, A., Remy, E., Thieffry, D.: Petri net representation of multi-valued logical regulatory graphs. Natural Computing **10**(2), 727–750 (2011)
12. Chaouiya, C., Remy, E., Ruet, P., Thieffry, D.: Qualitative modelling of genetic networks: From logical regulatory graphs to standard Petri nets. In: J. Cortadella, W. Reisig (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)
13. Chatain, T., Haar, S., Jezequel, L., Paulevé, L., Schwoon, S.: Characterization of reachable attractors using Petri net unfoldings. In: Computational Methods in Systems Biology, *Lecture Notes in Computer Science*, vol. 8859, pp. 129–142. Springer (2014)
14. Chatain, T., Haar, S., Koutny, M., Schwoon, S.: Non-atomic transition firing in contextual nets. In: Applications and Theory of Petri Nets, *Lecture Notes in Computer Science*, vol. 9115, pp. 117–136. Springer (2015). DOI 10.1007/978-3-319-19488-2_6
15. Chatain, T., Haar, S., Paulevé, L.: Boolean Networks: Beyond Generalized Asynchronicity. In: J.M. Baetens, M. Kutrib (eds.) Cellular Automata and Discrete Complex Systems (AUTOMATA 2018), *Lecture Notes in Computer Science*, vol. 10875, pp. 29–42. Springer, Ghent, Belgium (2018)
16. Chatain, T., Paulevé, L.: Goal-Driven Unfolding of Petri Nets. In: R. Meyer, U. Nestmann (eds.) 28th International Conference on Concurrency Theory (CONCUR 2017), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 85, pp. 18:1–18:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). DOI 10.4230/LIPIcs.CONCUR.2017.18
17. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. Theoretical Computer Science **147**(1&2), 117–136 (1995). DOI 10.1016/0304-3975(94)00231-7
18. Collombet, S., van Oevelen, C., Sardina Ortega, J.L., Abou-Jaoudé, W., Di Stefano, B., Thomas-Chollier, M., Graf, T., Thieffry, D.: Logical modeling of lymphoid and myeloid cell specification and transdifferentiation. Proc. Natl. Acad. Sci. **114**(23), 5792–5799 (2017). DOI 10.1073/pnas.1610622114
19. Courtiat, J., Saïdouni, D.: Relating maximality-based semantics to action refinement in process algebras. In: Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques, Berne, Switzerland, 1994, *IFIP Conference Proceedings*, vol. 6, pp. 293–308. Chapman & Hall (1995)
20. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Springer (2008)
21. Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., De Micheli, G.: Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics **24**(17), 1917–1925 (2008). DOI 10.1093/bioinformatics/btn336
22. Goss, P.J.E., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proceedings of the National Academy of Sciences **95**(12), 6750–6755 (1998). DOI 10.1073/pnas.95.12.6750
23. Janicki, R., Koutny, M.: Structure of concurrency. Theoretical Computer Science **112**(1), 5–52 (1993). DOI 10.1016/0304-3975(93)90238-O
24. Janicki, R., Koutny, M.: Fundamentals of modelling concurrency using discrete relational structures. Acta Inf. **34**, 367–388 (1997)
25. Janicki, R., Lauer, P.E., Koutny, M., Devillers, R.R.: Concurrent and maximally concurrent evolution of nonsequential systems. Theor. Comput. Sci. **43**, 213–238 (1986). DOI 10.1016/0304-3975(86)90177-5. URL https://doi.org/10.1016/0304-3975(86)90177-5
26. Kauffman, S.A.: Metabolic stability and epigenesis in randomly connected nets. Journal of Theoretical Biology **22**, 437–467 (1969). DOI 10.1016/0022-5193(69)90015-0
27. Kolčák, J., Šafránek, D., Haar, S., Paulevé, L.: Parameter Space Abstraction and Unfolding Semantics of Discrete Regulatory Networks. Theoretical Computer Science (2018). In press
28. Mai, Z., Liu, H.: Boolean network-based analysis of the apoptosis network: Irreversible apoptosis and stable surviving. Journal of Theoretical Biology **259**(4), 760 – 769 (2009). DOI https://doi.org/10.1016/j.jtbi.2009.04.024
29. Martínez-Sosa, P., Mendoza, L.: The regulatory network that controls the differentiation of t lymphocytes. Biosystems **113**(2), 96 – 103 (2013). DOI https://doi.org/10.1016/j.biosystems.2013.05.007

30. Noual, M., Sené, S.: Synchronism versus asynchronism in monotonic boolean automata networks. Natural Computing (2017). DOI 10.1007/s11047-016-9608-8
31. Palma, E., Salinas, L., Aracena, J.: Enumeration and extension of non-equivalent deterministic update schedules in boolean networks. Bioinformatics **32**(5), 722–729 (2016). DOI 10.1093/bioinformatics/btv628
32. Paulevé, L.: Reduction of Qualitative Models of Biological Networks for Transient Dynamics Analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics (2017). DOI 10.1109/TCBB.2017.2749225. In press
33. Popova-Zeugmann, L., Heiner, M., Koch, I.: Time Petri nets for modelling and analysis of biochemical networks. Fundamenta Informaticae **67**(1), 149–162 (2005)
34. Remy, E., Ruet, P., Thieffry, D.: Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. Advances in Applied Mathematics **41**(3), 335 – 350 (2008). DOI 10.1016/j.aam.2007.11.003
35. Richard, A.: Negative circuits and sustained oscillations in asynchronous automata networks. Advances in Applied Mathematics **44**(4), 378 – 392 (2010). DOI 10.1016/j.aam.2009.11.011
36. Rodríguez, C., Schwoon, S.: Cunf: A tool for unfolding and verifying Petri nets with read arcs. In: International Symposium on Automated Technology for Verification and Analysis, pp. 492–495. Springer (2013)
37. Rougny, A., Froidevaux, C., Calzone, L., Paulevé, L.: Qualitative dynamics semantics for SBGN process description. BMC Systems Biology **10**(1), 1–24 (2016). DOI 10.1186/s12918-016-0285-0
38. Schwoon, S.: Mole. http://www.lsv.ens-cachan.fr/ schwoon/tools/mole/
39. Schönfisch, B., de Roos, A.: Synchronous and asynchronous updating in cellular automata. Biosystems **51**(3), 123 – 143 (1999). DOI 10.1016/S0303-2647(99)00025-8
40. Steggles, L.J., Banks, R., Shaw, O., Wipat, A.: Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach. Bioinformatics **23**(3), 336–343 (2007). DOI 10.1093/bioinformatics/btl596
41. Thieffry, D., Thomas, R.: Dynamical behaviour of biological regulatory networks – II. Immunity control in bacteriophage lambda. Bulletin of Mathematical Biology **57**, 277–297 (1995). DOI 10.1007/BF02460619
42. Thomas, R.: Boolean formalization of genetic control circuits. Journal of Theoretical Biology **42**(3), 563 – 585 (1973). DOI 10.1016/0022-5193(73)90247-6
43. Traynard, P., Fauré, A., Fages, F., Thieffry, D.: Logical model specification aided by model-checking techniques: application to the mammalian cell cycle regulation. Bioinformatics **32**(17), i772–i780 (2016). DOI 10.1093/bioinformatics/btw457
44. Vogler, W.: Fairness and partial order semantics. Inf. Process. Lett. **55**(1), 33–39 (1995). DOI 10.1016/0020-0190(95)00049-I
45. Vogler, W.: Partial order semantics and read arcs. Theoretical Computer Science **286**(1), 33–63 (2002)
46. Winkowski, J.: Processes of contextual nets and their characteristics. Fundamenta Informaticae **36**(1) (1998)