Model Checking Reversible Systems: Forwardly ⋆

Federico Dal Pio Luogo, Claudio Antares Mezzina, G. Michele Pinna

Model Checking Reversible Systems: Forwardly*

Federico Dal Pio Luogo[1], Claudio Antares Mezzina[0000−0003−1556−2623][2], and G. Michele Pinna[0000−0001−8911−1580][1]

[1] Dipartimento di Matematica e Informatica, Università di Cagliari, Italy
[2] Dipartimento di Scienze Pure e Applicate, Università di Urbino, Urbino, Italy

Abstract. Reversibility is nowadays playing a major role when dealing with systems, allowing to revert to safe states of systems evolutions. For instance reversibility can be applied to causal-consistent debugging. On the other hand, Linear Temporal Logic (LTL) has been used to formalize properties that a system may fulfil, and it may be equipped with past operators. This makes this logic appealing to express and prove properties of a reversible system. In this paper we investigate this feature, and we use the classical approaches to model check LTL formulas on unfoldings, in order to deal with reversible systems.

1 Introduction

Reversibility has attracted interests since the 70's for its promise of reaching low-energy computations. Recently, reversibility has found applications in modelling biochemical reactions [28, 26], enhancing parallel discrete event simulators [29], formalising fault-tolerant systems [3, 13, 31, 22], and improving reversible debuggers [10, 16]. Due to this versatility, several classical formalisms have been adapted to model reversible systems: reversible process calculi [2, 27, 14], reversible Petri nets [21, 20, 23] and reversible event structures [30, 19], just to name a few. Although many formalisms have emerged for modelling reversible computations, especially in concurrency, the verification of such systems has received little attention mainly focusing on equivalence theory [24, 25, 15].

Model checking is a well-know verification technique. It involves building a finite state model of the system to be verified, and then proving that desired properties, expressed as logic formulae, hold on the model. Usually properties are expressed using a temporal logic, for instance *linear temporal logic* (LTL). Such a logic has modal operators to deal with the future of events, and its

* This work was supported by the Project PRIN 2022 DeLiCE (F53D23009130001) funded under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, by the Italian MUR PRIN 2020 project NiRvAna, by the Italian Ministry of Education, University and Research through the PRIN 2022 project "Developing Kleene Logics and their Applications" (DeKLA), project code: 2022SM4XC8, and the European Union - NextGenerationEU SEcurity and RIghts in the CyberSpace (SERICS) Research and Innovation Program PE00000014, projects STRIDE and SWOP.

Main	**Thread A**	**Thread B**
...	...	
$x = 1;$	$a = x;$	$b = x;$
...	...	...

Fig. 1: A shared memory scenario involving two threads

applications to forward-only systems have been studied thoroughly (e.g. [1, 5] and quotation herein).

In a reversible system we can observe two directions of computation: the classical forward one, and a backward one aiming at undoing the effects of the forward computation. Hence, a natural question arises: how can we model check a reversible system? One option is to equip LTL with modal operators dealing with the past. In fact LTL can be endowed with past modal operators and the interesting result is that this extension does not add expressivity to LTL ([17]). Therefore a logic with past operators seems promising in the capability of describing the two computational directions of a reversible systems.

However the act of *undoing* the effect of a forward computational step can be seen itself as a forward action. This observation suggests that one can use just a *future* temporal logic only. Since LTL with past (PLTL) and LTL without past have the same expressive power, and there are various ways to encode PLTL into LTL (see for example [8]), we can just focus on LTL. The choice of considering LTL instead of other logics is also motivated by the fact that in the reversible system we are not interested in branching time, but rather on the linear line on which we can go also backward.

Consider the following shared memory scenario, depicted in Figure 1, in which two threads read the value of a variable shared with the main method: the threads can read the initial value of the variable, which is 0, or 1 depending on whether they have read the value of x before or after the main method has modified it.

Assuming sequential consistency, if we were to reverse debug such a program we would like to check the following property: if thread A reads the value 1 this means that it has never read 0 before, or it has read in the past 0 but then it has undone this action. This is a kind of property that can be expressed in a temporal logic with past operators. However, if we think in terms of future we could rephrase the above property as follows. If the thread A reads 1 then two situations are possible: either it will never read 0 or it can read 0 and then it will have undone this read. The above scenario can be described using *future* operators only.

We now turn our attention on how to describe the forward and backward computations of a reversible systems. A concurrent system can be seen as a product of several components that run in parallel and synchronise on some actions. Such components run sequentially without internal concurrency, but they exhibit internal nondeterminism due to some choices. This model of concurrency is not different from the real world ones. Take for example the actor-model (made popular by Erlang) where actors are sequential computational units and they

can synchronise with each other by message exchange. Hence, we consider our system as a *multi-clock net* (*mcn*-net), introduced in [6], which precisely captures the idea of single-threaded components running in parallel.

The choice of modelling a system with *mcn*-nets allows us to describe their behaviours with unfoldings that in this particular case turn out to be *mcn*-nets as well. Also, the unfolding produces an occurrence net, and we can reason about reversibility as it is done in [21], by simply adding for every transition we want to reverse an exact opposite one (e.g., consuming tokens in the postset and producing tokens in the preset). Despite these transitions, the unfolding remains a product of state machines without concurrency, or transition systems. Therefore, the final result is still a transition system, and we can use standard techniques to verify LTL formulae, as in [5].

We sum up our approach to model check forwardly properties regarding reversible computations of a reversible system:

1. we describe a concurrent system as a parallel composition of sequential processes,
2. we unfold the system to obtain a representation of all its possible behaviours,
3. we add reversible transitions to the unfolded system following [21], which can be *unfolded* again to obtain a *forward* representation of all the possible behaviours of a reversible system, and
4. we use standard LTL model checking to model check the obtained system.

Structure of the paper: We start, in the next section, by revising nets, multi-clock nets and unfoldings, and in Section 3 we describe the temporal logic we will use. In Section 4 we show how to add reversibility to the transition systems and in Section 5 we describe how properties of our reversible systems can be verified. Finally we draw some conclusions.

2 Petri nets, Multi-clock nets and Unfoldings

A concurrent system can be modelled by a suitable class of Petri nets, and its computations can be still expressed as a Petri net. Hence in this section, we will first introduce the class of Petri nets we will be using to model a concurrent system and then show how it is possible to express its behavious.

With $\mathbb{N}$ we denote the set of natural numbers. Let X be a set, with $|X|$ we denote the cardinality of the set.

We start reviewing the notions of (safe) labeled Petri nets and of token game. We fix a set Σ of transition names.

Definition 1. *A labeled* Petri net *over Σ is a 5-tuple $N = \langle S, T, F, m, \ell \rangle$, where S is a set of* places *and T is a set of* transitions *(with $S \cap T = \emptyset$), $F \subseteq (S \times T) \cup (T \times S)$ is the* flow *relation, $m \colon S \to \mathbb{N}$ is the* initial marking*, and $\ell \colon T \to \Sigma$ is a labeling mapping.*

Ordinary Petri nets are those where the labeling is the identity. Subscripts or superscript on the net name carry over the names of the net components. Given $x \in T \cup S$, ${}^\bullet x = \{y \mid (y,x) \in F\}$ and and $x^\bullet = \{y \mid (x,y) \in F\}$. ${}^\bullet x$ and $x^\bullet$ are called the *preset* and *postset* respectively of x. A net $\langle S, T, F, m, \ell \rangle$ is as usual graphically represented as a bipartite directed graph where the nodes are the places and the transitions, and where an arc connects a place s to a transition t iff $(s,t) \in F$ and an arc connects a transition t to a place s iff $F(t,s) \in F$. We assume that all nets we consider are such that $\forall t \in T$ ${}^\bullet t$ and $t^\bullet$ are not empty.

A transition t is enabled at a marking m, if m *contains* the pre-set of t, where contain here means that $m(s) \geq 1$ for all $s \in {}^\bullet t$. If a transition t is enabled at a marking m it may *fire*, yielding to a new marking m' defined as $m'(s) = m(s) - |{}^\bullet t \cap \{s\}| + |t^\bullet \cap \{s\}|$. The firing of t at m giving m' is denoted as $m\,[t\rangle\,m'$ and with $m\,[t\rangle$ we say that t is enabled at m. A marking $m' : S \to \mathbb{N}$ is *reachable* whenever there exists a sequence of transitions $t_0, \ldots, t_n$ and markings $m_0, \ldots, m_{n+1}$ such that $m_i\,[t_i\rangle\,m_{i+1}$ for all $i \leq n$, m_0 is the initial marking and m_{n+1} is m'. The set of reachable markings of a net N is denoted with $\mathcal{M}_N$. A net N is said to be *safe* whenever its places hold at most one token in all possible evolutions, namely $\forall m \in \mathcal{M}_N$ it holds that m can be seen as a set (the only possible values are 0 and 1). We will consider safe nets only, and we will identify markings with the subsets of places carrying a token.

We recall some notions that will come in handy in the following.

State machines: A state machine net is a safe net where only choices are allowed, and at each reachable marking if two or more transitions are enabled, then only one can fire. Formally:

Definition 2. *Let $N = \langle S, T, F, m, \ell \rangle$ be a safe net. N is said to be a* state machine net *whenever $\forall t \in T$ it holds that $|{}^\bullet t| = 1 = |t^\bullet|$, and $\forall m' \in \mathcal{M}_N$ it holds that for all $t, t' \in T$ if $m'\,[t\rangle$ and $m'\,[t'\rangle$ then ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$.*

Net morphisms: The notion of morphism between safe nets [32] formalizes how labeled safe nets are related.

Definition 3. *Let $N = \langle S, T, F, m, \ell, \Sigma \rangle$ and $N' = \langle S', T', F', m', \ell', \Sigma' \rangle$ be safe nets over Σ and Σ' respectively. A* morphism *$\phi : N \to N'$ is a pair $\langle \phi_T, \phi_S \rangle$, where $\phi_T : T \to T'$ is a partial function and $\phi_S \subseteq S \times S'$ is a relation such that*

- *for each $s' \in m'$ there exists a unique $s \in m$ and $s\ \phi_S\ s'$,*
- *if $s\ \phi_S\ s'$ then the restriction $\phi_T\colon {}^\bullet s \to {}^\bullet s'$ and $\phi_T\colon s^\bullet \to s'^\bullet$ are total functions,*
- *if $t' = \phi_T(t)$ then $\phi_S^{op}\colon {}^\bullet t' \to {}^\bullet t$ and $\phi_S^{op}\colon t'^\bullet \to t^\bullet$ are total functions, where ϕ_S^{op} is the opposite relation to ϕ_S,*
- *ϕ_T preserves labels, i.e. $\ell'(\phi_T(t)) = \ell(t)$.*

Morphisms among safe nets preserve reachable markings. Consider $\phi\colon N \to N'$, then for each $m, m' \in \mathcal{M}_N$ and transition t, if $m\,[t\rangle\,m'$ then $\phi_S(m)\,[\phi_T(t)\rangle\,\phi_S(m')$ where $\phi_S(m) = \{s' \in S' \mid \exists p \in m \text{ and } p\ \phi_S\ s'\}$.

Restriction of a net: A subnet is obtained by restricting places and transitions, and accordingly the flow relation and the initial marking.

Definition 4. *Let $N = \langle S, T, F, m, \ell \rangle$ be a labeled Petri net and let $S' \subseteq S$ be a subset of places and $T' = {}^\bullet S' \cup S'^\bullet$. The subnet generated by S', denoted with $N|_{S'}$ is the net $\langle S', T', F', m', \ell' \rangle$, where F' is the restriction of F to S' and T', m' is the multiset on S' obtained by m restricting to the places in S' and ℓ' is the restriction of ℓ to the transitions in T'.*

Product of safe nets: We define how to combine components represented as safe nets. The idea is that the components may *share* transitions with the same labels, synchronising on them. We assume w.l.o.g. that nets have disjoint places and transitions.

Definition 5. *Given two safe nets $N = \langle S, T, F, m, \ell \rangle$ and $N' = \langle S', T', F', m', \ell' \rangle$ on Σ and Σ' respectively, such that $S \cap S' = \emptyset$ and $T \cap T' = \emptyset$, their* product *is defined as the safe net $N \times N' = \langle S \cup S', \hat{T}, \hat{F}, m + m', \hat{\ell} \rangle$ on $\Sigma \cup \Sigma'$ where*

- $\hat{T} = T \setminus \{t \in T \mid \ell(t) \in \ell'(T')\} \cup T' \setminus \{t' \in T' \mid \ell'(t') \in \ell(T)\} \cup \{\{t\} \cup \{t'\} \mid t \in T, t' \in T' \ \wedge \ \ell(t) = \ell'(t')\}$,
- $(s, \{x, y\}) \in \hat{F}$ *iff* $(s, x) \in F$ *or* $(s, y) \in F'$, $(\{x, y\}, s) \in \hat{F}$ *iff* $(x, s) \in F$ *or* $(y, s) \in F'$, *and*
- $\hat{\ell}(t) = \ell(t)$ *if* $t \in T \setminus \{t \in T \mid \ell(t) \in \ell'(T')\}$, $\hat{\ell}(t) = \ell'(t)$ *if* $t \in T' \setminus \{t' \in T' \mid \ell'(t') \in \ell(T)\}$ *and* $\hat{\ell}(\{t, t'\}) = \ell(t) = \ell'(t')$.

The projections from $N \times N'$ to N and N' are defined in the obvious way: the places of each component are related with the same places in the product and the mapping on transitions is undefined for the transitions of the other component, and it is the identity on the ones of the component.

We observe that the product on safe nets is commutative and associative.

Proposition 1. *The product of safe nets is associative and commutative, i.e. $N \times N' = N' \times N$ and $N \times (N' \times N'') = (N \times N') \times N''$.*

2.1 Multi-clock nets

Safe nets can be seen as formed by various *sequential* components (automata) synchronizing on common transitions. This intuition is formalized in the notion of *multi-clock* nets, introduced by Fabre in [6].

Definition 6. *A* multi-clock *net (mcn-net)* N *is the pair (N, ν) where $N = \langle S, T, F, m, \ell \rangle$ is a safe net and $\nu : S \to m$ is a mapping such that*

- *for all $s, s' \in m$, it holds that $s \neq s'$ implies $\nu^{-1}(s) \cap \nu^{-1}(s') = \emptyset$,*
- $\bigcup_{s \in m} \nu^{-1}(s) = S$,
- *ν is the identity when restricted to m, and*
- *for all $t \in T$. ν is injective on ${}^\bullet t$ and on $t^\bullet$, and $\nu({}^\bullet t) = \nu(t^\bullet)$.*

The cardinality *of a mcn-net* N*, denoted with* $\upsilon(\mathsf{N})$*, is the cardinality of* m*.*

Given $s \in S$, with $\overline{s}$ we denote the subset of places defined by $\nu^{-1}(\nu(s))$. The consequence of the last two requirements, namely (i) $\nu(m) = m$, (ii) ν is injective on the preset (postset) of each transition and that $\nu({}^\bullet t) = \nu(t^\bullet)$, is that, for each $s \in m$, the net $\langle \overline{s}, T_{\overline{s}}, F_{\overline{s}}, \{s\}, \ell_{\overline{s}} \rangle$ is a net automaton, *i.e.* the preset and the postset of each transition has exactly one element, where $T_{\overline{s}}$ are the transitions of N such that $\forall t \in T_{\overline{s}}\ {}^\bullet t \cap \overline{s} \neq \emptyset$ and $t^\bullet \cap \overline{s} \neq \emptyset$, and $F_{\overline{s}}$ is the restriction of F to $\overline{s}$ and $T_{\overline{s}}$. Each place s in the initial marking can be identified with an index in $\{1, \ldots, \upsilon(\mathsf{N})\}$, hence we denote N_i as the net $\langle \overline{s}, T_{\overline{s}}, F_{\overline{s}}, \{s\}, \ell_{\overline{s}} \rangle$ where i is the *index* of s.

Proposition 2. *Let* (N, ν) *be a mcn-net, with* $N = \langle S, T, F, m, \ell \rangle$*. Then* $N|_{\overline{s}}$ *is a state machine net.*

State-machine nets can be considered as finite state automata, and the consequence of what stated above is that $N = \langle S, T, F, m \rangle$ can be seen as *formed* by the various components. More precisely N is the product of the nets N_s where $s \in m$ and each N_s is $\langle \overline{s}, T_{\overline{s}}, F_{\overline{s}}, m_{\overline{s}} \rangle$.

Proposition 3. *Let* (N, ν) *be a mcn-net, with* $N = \langle S, T, F, m, \ell \rangle$*. Then* $N = \times_{s \in m} N|_{\overline{s}}$.

Sometimes multi-clock nets will be identified with the underlying safe net $N = \langle S, T, F, m \rangle$ and the partition mapping will be denoted with $\nu(N)$. It should be stressed out that the partition is not unique.

We have now enough material to re-elaborate the shared memory example given in the Introduction, casting it as a multi clock net.

Example 1. Let us consider the following shared memory scenario:

Main	**Thread A**	**Thread B**	**Possible Values**
$\ldots$	$\ldots$	$\ldots$	$x = 1; a = 0; b = 0;$
$x = 1;$	$a = x;$	$b = x;$	$x = 1; a = 1; b = 0;$
$\ldots$	$\ldots$	$\ldots$	$x = 1; a = 0; b = 1;$
			$x = 1; a = 1; b = 1;$

where a global variable x is shared among two threads and the main program. We assume that x is initialized to 0. The two threads read the value of x and copy it to their local variables (respectively a and b), while the main program modifies the value of the variable x by assigning 1 to it. Under sequential consistency, we can have four possible combinations of values, depending on the scheduling: either the two threads read x before the main method changes its value or one of the threads reads 0 and the other reads 1, or they both perform the reads after that the value of x has been changed. The above scenario can be modelled by the mcm-net in the top part of Figure 2. We can see that as long as the main program does not overwrite the value of x the two threads can only read 0 independently. This is modelled by the pink and light green transitions, which are independent (occur in different components). The act of assigning 1 to x,

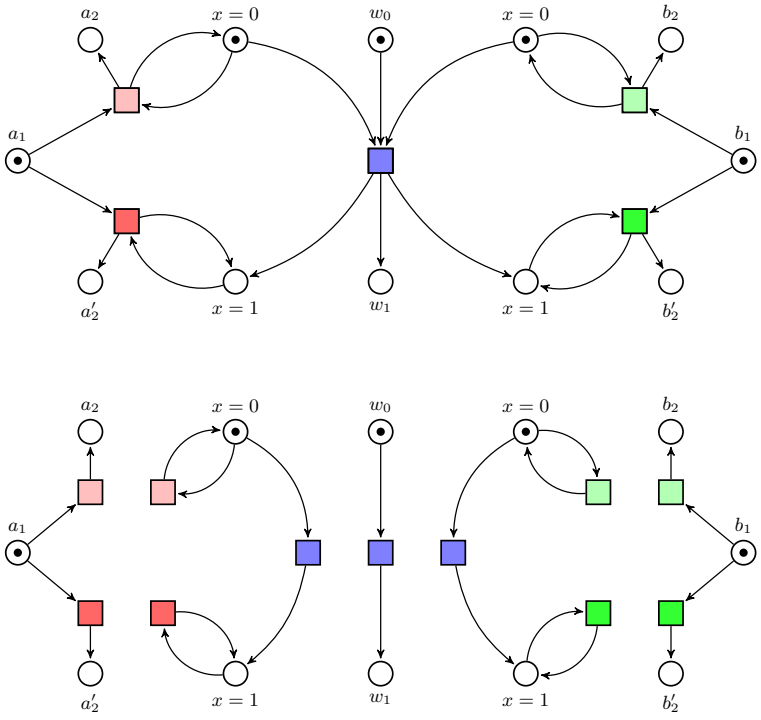


Fig. 2: An *mcn*-net and its components

by the main program, is rendered by the w transition which consumes the old value of the variable and produces the new value of the variable. Hence if the transition w takes place, the pink and light green transition cannot fire, while the red and green transition are enabled. These last transitions mimic the fact that the threads can potentially read 1, and they are modelled so that their execution is independent of the other. The bottom part of Figure 2 shows the state machine nets whose product is the mcm-net (according to Proposition 3).

When we turn our attention to *mcn*-nets, we consider morphisms that preserve the partitions.

Definition 7. *Let* $N = (\langle S, T, F, m, \ell\rangle, \nu)$ *and* $N' = (\langle S', T', F', m', \ell'\rangle, \nu')$ *be two multi-clock nets. A* morphism $\phi : N \to N'$ *is a mcn-morphism iff* $\forall s \in S$, $\forall s' \in S'$, $s\ \phi_S\ s'$ *implies that* $\nu(s)\ \phi_S\ \nu'(s')$.

2.2 Occurrence nets

Once we have modelled a system using (multi-clock) nets, we turn our attention on how to describe the behaviours of the system. And on such behaviours we will prove properties of the original system. The behaviour of a Petri net can be

described using a suitable net ([4, 32, 11] among others). The idea is that a net can be *unrolled*, yielding to an acyclic net called *occurrence net*, which describes all the possible executions of a net.

Let $N = \langle S, T, F, m, \ell \rangle$ be a labeled safe Petri net, the *causality relation* $<$ *is the transitive closure of* F *and* $\leq$ *is its reflexive transitive closure. The set of causes of a node* x *is defined as* $\lfloor x \rfloor = \{y \in T \cup S \mid y \leq x\}$. *Two different nodes* x *and* y *are in* conflict, written $x \# y$ if there are transitions t_1 and t_2 such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ and $t_1 \leq x$ and $t_2 \leq y$. Execution paths of the Petri net branch at the conditions where the conflict originates. Finally, the concurrency relation **co** holds between nodes $x, y \in S \cup T$ that are neither ordered nor in conflict, i.e. x **co** $y \Leftrightarrow \neg(x \leq y) \wedge \neg(y \leq x) \wedge \neg(x \# y)$.

Definition 8. *A Petri net* $N = \langle S, T, F, m, \ell \rangle$ *is an* occurrence net *whenever*

- $\leq$ *is a partial order;*
- *for all* $s \in S$, $|{}^\bullet s| \leq 1$*;*
- *for all* $x \in S \cup T$, *the set* $\lfloor x \rfloor$ *is finite;*
- *no node is in self-conflict, i.e. there is no* $x \in B \cup E$ *such that* $x \# x$*; and*
- *for all* $s \in m$. ${}^\bullet s = \emptyset$.

In literature, places and transitions of occurrence nets are called respectively *conditions* and *events* and denoted with B and E respectively. Therefore an occurrence net will be denoted with $C = \langle B, E, F, c \rangle$ with $c \subseteq B$ being the initial marking. The initial marking is such that the conditions $b \in B$ have the property that ${}^\bullet b = \emptyset$. The occurrence net associated to a safe net N, together with a way relating places and transitions of this occurrence net to those of the net N, is called *unfolding* of the net. The unfolding holds useful properties that can be exploited to make the semantic of a net a reversible one.

Definition 9. *Let* $N = \langle S, T, F, m, \ell \rangle$ *be a safe net on* Σ, $C = \langle B, E, F, c, \ell_C \rangle$ *be an occurrence net on* Σ *and let* $\phi = (\phi_S, \phi_T) : C \to N$ *be a net-morphism such that*

- $\forall e \in E$, $\ell_C(e) = \ell(\phi_T(e))$,
- ϕ_S^{-1} *and* $\phi_S|_m$ *are total functions; and*
- $\phi_S|_m$ *is injective.*

Then (C, ϕ) *is an* unfolding *of* N. *The morphism* ϕ *is the folding morphism.*

The construction of the unfolding, i.e. of the occurrence net and of the folding morphism, is standard and it is omitted here. When considering nets with cycles, as the unfolding represents all the computations, the resulting occurrence net may be infinite. However it is enough to consider, of the unfolding, only a finite part, provided that it is *complete*, namely all the possible computations are somehow represented (see [18, 12] among others). Here we consider unfoldings that are complete prefixes of an unfolding.

We now characterize better the unfolding obtained from a *mcn*-net.

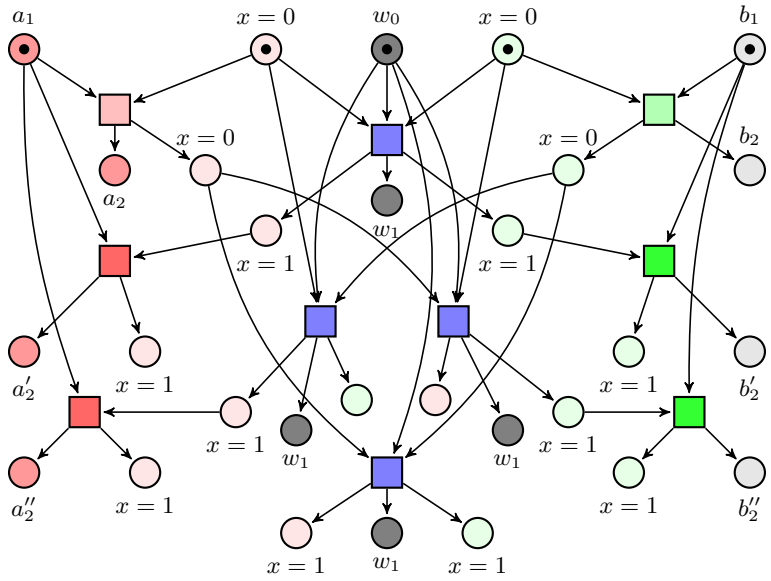


Fig. 3: The unfolding of the *mcn*-net in Figure 2. The conditions arising from the places of each state machine are coloured in the same way.

Proposition 4. *Let (N, ν) be an mcn-net on Σ, $C = \langle B, E, F, c, \ell_C \rangle$ be an occurrence net on Σ and $\phi : C \to N$ be a net-morphism such that (C, ϕ) is an unfolding of N. Define $\nu_C : B \to c$ as follows: $\nu_C(b) = b'$ when $\forall s, s' \in \phi_S^{-1}(b)$. $\nu(s) = \nu(s')$ and $\nu(s)\phi_S b'$. Then (C, ν_C) is an mcn-net.*

The relevant result is that the unfolding of an *mcn*-net, which is the product of some components, is an occurrence net which is still the product of some components.

Proposition 5. *Let (N, ν) be an mcn-net on Σ and (C, ϕ) be an unfolding of N, with $N = \langle S, T, F, m, \ell \rangle$ and $C = \langle B, E, F, c, \ell_C \rangle$. Then $C = \times_{b \in c} C|_{\overline{b}}$.*

Example 2. Figure 3 depicts the complete unfolding of the *mcn*-net in the Example 1. Observe that there are four writes (depending on whether or not $x = 0$ has been read by one of the readers) and each reader can read $x = 1$ in two different ways. When restricting the net to various components we have the various unfoldings that keep memory of the previous interactions.

3 Past Linear Temporal Logic

Past Linear Temporal Logic (PLTL) is a *modal temporal logic* that extends propositional logic with time operators (formally, modalities) to reason on (the

future or the past of) paths of events. In the context of system verification, the atomic propositions of the formulae are state labels, or basic assertions about the states of the system under consideration.

Syntax: We fix a set AP of atomic propositions. A PLTL formula, ranged over by Greeks letters, obeys to the grammar

$$\psi ::= \text{true} \mid a \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \bigcirc \psi \mid \odot \psi \mid \psi_1 \,\mathsf{U}\, \psi_2 \mid \psi_1 \,\mathsf{S}\, \psi_2$$

where $a \in AP$. The temporal modalities are $\bigcirc$ (pronounced *next*), $\odot$ (pronounced *previous*), U (pronounced *until*) and S (pronounced *since*). We indicate with $PLTL$ the set of all the possible formulae generated by the above grammar. The until operator allows to derive the future temporal modalities $\Diamond$ (*eventually*, sometime in the future) and $\Box$ (*always* in the future, from now on forever) as follows $\Diamond \psi \stackrel{\text{def}}{=} \text{true}\,\mathsf{U}\,\psi$ and $\Box \psi \stackrel{\text{def}}{=} \neg\Diamond\neg\psi$, whereas the since operator allows to derive the temporal modalities $\blacklozenge$ (*once*, sometime in the past) and $\boxdot$ (*always* in the past), hence $\blacklozenge\psi \stackrel{\text{def}}{=} \text{true}\,\mathsf{S}\,\psi$ and $\boxdot\,\psi \stackrel{\text{def}}{=} \neg\blacklozenge\neg\psi$.

Semantics: PLTL views time as a discrete domain, whose *ticks* (advances of a single time unit) correspond to transitions of the system under observation. The present refers to the current state (e.g., state i) and the next moment corresponds to the immediate successor state (e.g., $i+1$). The system behavior is assumed to be observable at the time points $0, 1, 2, \ldots$.

The semantics of PLTL is defined over infinite sequences of sets of atomic propositions. Intuitively, these sets correspond to the conditions that the system satisfies in a given time frame. The occurrence of a transition triggers a change in the state of the system and consequently in the basic assertions (i.e. atomic propositions) that hold in the next frame. The i-th letter $A_i \subseteq AP$ of a word $\sigma = A_0 A_1 A_2 \ldots$ corresponds to the assertions that hold at the i-th time frame, where $i \geq 0$. With (σ, j) we denote the suffix of σ starting at index $j \geq 0$.

Definition 10 (Semantics of PLTL). *The relation* $\models \subseteq (2^{AP})^\omega \times \mathbb{N} \times PLTL$ *is defined inductively as follows:*

$(\sigma, i) \models \text{true}$	always	
$(\sigma, i) \models a$	iff	$a \in A_i$
$(\sigma, i) \models \psi_1 \wedge \psi_2$	iff	$(\sigma, i) \models \psi_1$ *and* $(\sigma, i) \models \psi_2$
$(\sigma, i) \models \neg\psi$	iff	$(\sigma, i) \not\models \psi$
$(\sigma, i) \models \bigcirc \psi$	iff	$(\sigma, i+1) \models \psi$
$(\sigma, i) \models \odot \psi$	iff	$i > 0$ *and* $(\sigma, i-1) \models \psi$
$(\sigma, i) \models \psi_1 \,\mathsf{U}\, \psi_2$	iff	$\exists j \geq i\ (\sigma, j) \models \psi_2$ and $\forall k.\ i \leq k < j\ (\sigma, k) \models \psi_1$
$(\sigma, i) \models \psi_1 \,\mathsf{S}\, \psi_2$	iff	$\exists j.\ 0 \leq j \leq i.\ (\sigma, j) \models \psi_2$ and $\forall k.\ j < k \leq i\ (\sigma, k) \models \psi_1$

The PLTL *property* induced by the formula ψ (or equivalently the *model of* ψ), denoted by $Words(\psi)$, is the language comprising the infinite words that satisfy the property: $Words(\psi) = \left\{\sigma \in (2^{AP})^\omega \mid (\sigma, 0) \models \psi\right\}$

Example 3. Consider the propositional letters $AP = \{a, b\}$ and the PLTL formula $\psi = \Diamond\,(\diamondsuit\, a \wedge \bigcirc\, b)$ which says simply that there will be a point where b holds and before that in a point a held. The word $\sigma = \emptyset^n\{a\}\emptyset^m\{b\}\ldots$ is such that $(\sigma, 0) \models \psi$. Observe the ψ can be written using forward modalities only as $\phi = \Diamond\,(a \wedge \Diamond\, \bigcirc\, b)$, as it is easy to observe that $\forall\sigma \in (2^{AP})^\omega$ it holds that $(\sigma, 0) \models \psi$ iff $(\sigma, 0) \models \phi$, hence $Words(\psi) = Words(\phi)$.

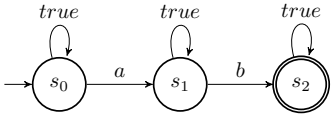
3.1 Modeling a (P)LTL formula and model checking it

According to [17] past modalities do not add expressive power to LTL, but are useful to express more succinctly formulae. Hence, any formula with past modalities can be rewritten into one using just forward modalities as done for example in [7, 8]. Therefore, we will be considering just forward LTL formulae.

We recall that the model of any LTL formula ψ can be represented as the language accepted by a nondeterministic Büchi automaton (NBA), which is a nondeterministic finite state automaton where accepting states have to be visited infinitely often. Given an NBA $\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ the language recognized by it is denoted as $\mathcal{L}^\omega(\mathcal{A})$ and it is the set of infinite words $A_0A_1A_2\ldots$ such that there exists an infinite sequence of states $q_0q_1q_2\ldots$ such that $\forall i \in \mathbb{N}$. $q_{i+1} \in \delta(g_i, A_i)$ with $q_0 \in Q_0$ and there exists infinite indexes j such that $q_j \in F$.

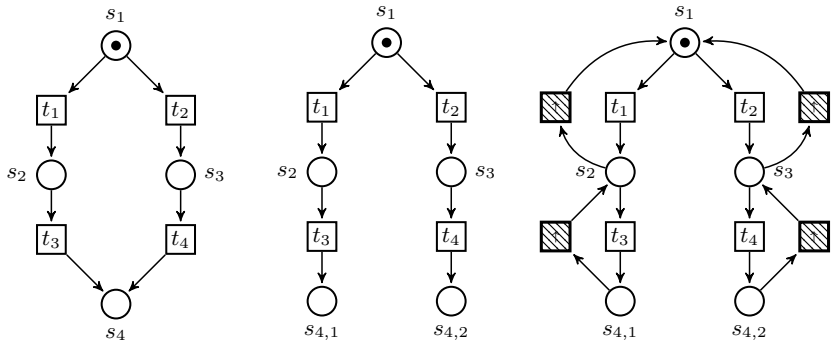
Proposition 6. *Given an LTL formula ψ, there exists an NBA automaton $\mathcal{A}_\psi$ such that $\mathcal{L}^\omega(\mathcal{A}_\psi) = Words(\psi)$.*

Example 4. The following NBA accepts the words $\sigma \in (2^{\{a,b\}})^\omega$ modeling the formulae in Example 3:



Once that the automaton is built, it is sufficient to build a *product* of the automaton with the representation of the behaviours of the system (on which we want to verify a formula) and then check whether there exists a way to fulfil the accepting conditions of $\mathcal{A}$. If this is the case, then the formula is satisfied by the system.

The unfolding of a multi-clock net can be seen as the product of components, hence given a multi-clock net, we can compute its unfolding, and then construct the product of the unfolding with the automaton $\mathcal{A}$. This is possible since the unfolding of a *mcn*-net can be seen as the product of several state machines (see Property 5), and the automaton itself is a state machine.



(a) N (b) C_1 (c) C_2

Fig. 4: A net with backward conflict (left), its unfolding (center) and its reversible version (right).

4 How to model Reversibility

A reversible computational model is a model which is able of describing not only the forward direction of the computation but also the backward one. Starting from a system modeled as *mcn*-net we are able to represent its forward behaviour as an unfolding. It is worth recalling that unfolding a *mcn*-net produces an occurrence net which is still a *mcn*-net. For example, if we take the unfolding of our running example, which is depicted in Figure 3 it is easy to check that the net is an occurrence one, where the initial markings is represented by all the places (conditions) with no incoming arc. One important property of the unfolding is the absence of *backward conflicts* (e.g., a token in a place can be produced by two or more transitions). Backward conflicts are not desirable in reversibility because they do not allow to distinguish between transitions putting a token in the same place. Graphically, a backward conflict is represented by a place with two or more input edges. The unfolding removes backward conflicts by creating a copy of this state for each of its input transitions, explicitly rendering the different execution paths that lead to it. Hence, for each place it is always possible to know which transition generated a token in it.

Since all the places (or conditions) have a unique path, adding reversibility is quite straightforward: to undo any transition, we simply add its *inverse*, where the postset of the original transition becomes the preset of the undoing transition, and vice versa. For example, let us consider the net N on the left of Figure 4. If we have a token in the place s_4, we really cannot tell whether the token has been produced by transition t_3 (e.g., left branch) or by transition t_4 (e.g., right branch). By unfolding the net on the left of Figure 4, we obtain the net C_1 on the center, where now the place s_4 has been duplicated into two places: each one having a unique history. Now, the resulting net can be easily *reversed*, by just adding for each transition we want to reverse a transition which is the exact inverse. For example, the inverse of transition t_1, consumes the token produced

by t_1 and puts it back. Once we have an unfolding, adding reversible semantics to it is easy: we equip the unfolding with as many *undo* events as there are events that are labeled by a transition we wish to undo. In [21] it is shown how it is possible to obtain a causal-consistent reversible semantics of a Petri net via its unfolding into occurrence nets: for every transition that one wants to undo it is just sufficient to add the exact opposite transition where each undo event has the postset of the original event as its preset, and the preset of it as its postset. This transformation introduces cycles in the unfolding - any execution could execute and immediately undo the same event infinitely often, therefore the resulting net cannot be considered an occurrence net. These cycles are necessary in the automata approach to model checking.

The above intuition is formalized in the notion of reversible net which is a Petri net where some transition *perform* the step of the backward flow (the reversing transitions) and removing these ones we have an occurrence net, which is related to the original system under consideration via a folding morphism.

Definition 11. *Let $N = \langle S, T, F, m\rangle$ be a net and let $\underline{T} \subseteq T$ be a subset of transitions such that*

- *$\forall \underline{t} \in \underline{T}$ there exists a unique $t \in T \setminus \underline{T}$ with ${}^\bullet t = \underline{t}^\bullet$ and $t^\bullet = {}^\bullet\underline{t}$;*
- *$\forall t, t', t'' \in T$, it ${}^\bullet t = t'^\bullet = t''^\bullet$ and $t^\bullet = {}^\bullet t' = {}^\bullet t''$ then $t' = t''$; and*
- *$\langle S, \overline{T}, \overline{F}, m\rangle$ is an occurrence net, where $\overline{T} = T \setminus \underline{T}$ and $\overline{F} = F \cap ((S \times \overline{T}) \cup (\overline{T} \times S))$.*

Then $(N, \underline{T})$ is a reversing *net with $\underline{T}$ as reversing transitions.*

The first condition stipulates that for each reversing transition there is just one forward one, and the second says that the set of forwards transitions having a reverse is bijectively related with the set of reversing transitions.

Operatively, given an occurrence net C, we can add reversing transition as follows.

Definition 12. *Let $C = \langle B, E, F, c\rangle$ be an occurrence net and let $E' \subseteq E$ be a subset of transitions to be reversed. Let $\overleftarrow{C} = \langle B, E \cup \overleftarrow{E'}, \overleftarrow{F}, c\rangle$ where $\overleftarrow{E'} = \{(e, \mathtt{r}) \mid e \in E'\}$ and $\overleftarrow{F} = F \cup \{(b, (e, \mathtt{r})) \mid (e, b) \in F\} \cup \{((e, \mathtt{r}), b) \mid (b, e) \in F\}$. Then $(\overleftarrow{C}, \overleftarrow{E'})$ is a reversing net with $\overleftarrow{E'}$ as reversing transitions.*

We simply add the reversing transitions that have the effect of undoing the forward one.

Example 5. Given the net N in Figure 4, the net C_1 is the unfolding of N into an occurrence net. The net C_2 is the reversing net of C_1, that is $C_2 = (\overleftarrow{C_1}, \{(t_1, \mathtt{r}), (t_2, \mathtt{r}), (t_3, \mathtt{r}), (t_4, \mathtt{r})\})$.

With this approach we add reversibility to our model.

Example 6. Figure 5 reports the reversible variant of Figure 3. For the sake of clarity, we have just added one backward transition (the one labelled with $\uparrow$), which corresponds to the undoing of the read of the thread A when it reads 0. In the model we have considered (and model checked) all the transitions have been reversed.

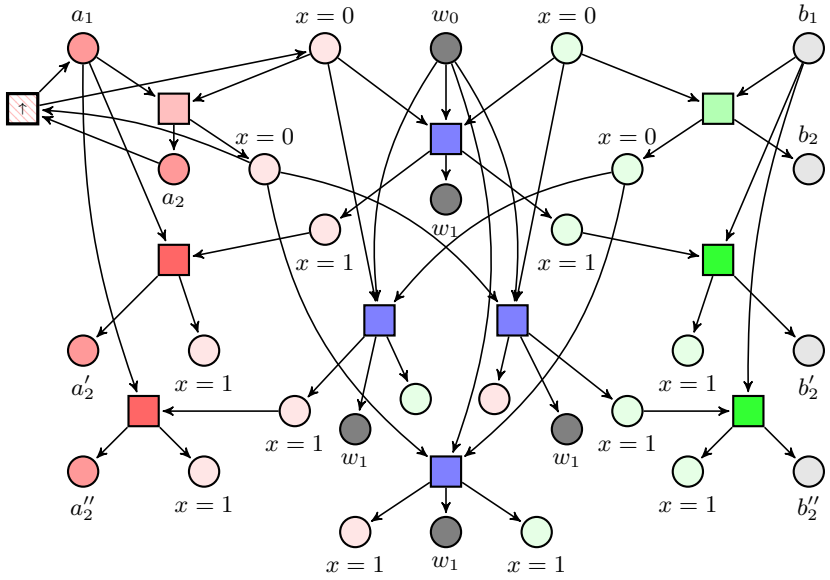


Fig. 5: Adding reversible transitions

Let us note that by adding reversing transitions to the unfold of a *mcn*-net, we still obtain an *mcn*-net, which is still a product of sequential components.

Proposition 7. *Let $((C, \nu_C), \phi)$ be an unfolding of an mcn-net (N, ν) and let $E' \subseteq E$ be a set of reversing transitions, then $\overleftarrow{C}$ is a reversing net and $(\overleftarrow{C}, \nu_C)$ is a mcn-net.*

We do not care too much about the morphism from the enriched unfolding to the original net as it is not important for our purposes. We just notice that the extension of the mapping on transitions which is undefined on the reversing transition would work perfectly.

By adding reversible transitions we obtain a net which is not any longer an occurrence net, it is however the model of the behaviour we are interested in, and on these we can reason about the properties we want to check.

5 Model checking a Formula

To prove that the behaviours of a system or program satisfies some property, a standard method is to use model checking. The method we use to perform model checking is very classical and it is known as the *automata-theoretic approach*. It can be summed up in three steps:

1. translate the negation of the formula to be checked into a Büchi automaton,

2. synchronize the system and the Büchi automaton in an adequate way to yield a composed system, and
3. check emptiness of the language of the composed system.

This approach works well with a *state-based* version of LTL, i.e. LTL where the atomic propositions are the states of the system under verification. We first introduce the details about model checking, and then we sketch reversibility properties using the states of the enriched unfolding and finally apply the model checking solution to handle reversibility semantics of arbitrary *mcn*-nets (we will often refer to these as *products*).

Let ψ be a formula of LTL. Using well known algorithms we can construct a Büchi automaton $\mathcal{A}_{\neg\psi}$ for the negation of ψ and build a tester for checking the emptiness of the language accepted by the compose system.

5.1 Constructing the tester to model check a formula

Let $\mathsf{N} = (N, \nu)$ be a *mcn*-net with $N = \langle S, T, F, m, \ell\rangle$ and let ψ be an LTL formula over S. The *model checking problem* consists of deciding whether $\mathsf{N} \models \psi$ holds.

We first observe that, given a *mcn*-net N, its unfolding represents a language that we denote as $Traces(\mathsf{N}) \subseteq (2^S)^\omega$, by simply observing that each element of a trace is indeed a reachable marking of the unfolding. Now, being N a multi-clock net, also its unfolding is a multi-clock net, which implies that in the reachable marking just one place of each component is marked. A Büchi automaton $\mathcal{A}_{\neg\psi}$ for the negation of the formula recognizes the language $Words(\neg\psi)$ containing all the infinite words that satisfy $\neg\psi$, i.e. $\forall\sigma \in Words(\neg\psi)$. $(\sigma, 0) \models \neg\psi$. The classic construction of the automaton for the LTL formula (e.g. see [1]) yields an NBA that is exponential in the size of the formula.

Therefore checking whether ψ holds for N reduces to verifying whether the language $Traces(\mathsf{N}) \cap Words(\neg\psi)$ is empty. Formally we have:

$$\begin{aligned}
\mathsf{N} \models \psi \quad &\text{iff} \quad Traces(\mathsf{N}) \subseteq Words(\psi) \\
&\text{iff} \quad Traces(\mathsf{N}) \cap ((2^S)^\omega \setminus Words(\psi)) = \emptyset \\
&\text{iff} \quad Traces(\mathsf{N}) \cap (Words(\neg\psi)) = \emptyset \\
&\text{iff} \quad Traces(\mathsf{N}) \cap (\mathcal{L}_\omega(\mathcal{A}_{\neg\psi})) = \emptyset
\end{aligned}$$

In order to provide an answer to the emptiness problem of $Traces(\mathsf{N}) \cap (\mathcal{L}_\omega(\mathcal{A}_{\neg\psi}))$, we need to construct a new device recognizing the intersection of the two languages. Such a device can be obtained by the synchronous product of N and $\mathcal{A}_{\neg\psi}$. Observe that any automaton can be seen as a state machine, hence the product we are going to use is well defined.

Definition 13. *Let $\mathsf{N} = (N, \nu)$ be a mcn-net with $N = \langle S, T, F, m, \ell\rangle$ and let $\mathcal{A}_{\neg\psi} = \langle S', T', F', m', \ell'\rangle$ be a state machine net encoding the Büchi automaton recognizing $\neg\psi$ labelled over $\Sigma = T$. The* full synchronization *of N and $\mathcal{A}_{\neg\psi}$ is the product $\mathsf{N} \times \mathcal{A}_{\neg\psi} = \langle S \cup S', \hat{T}, \hat{F}, m + m', \hat{\ell}\rangle$ given by Definition 5.*

In the full synchronization the Büchi tester participates pervasively in every transition of the product.

Therefore to check whether ψ holds for N one first constructs the NBA for the negation of the input formula ψ, representing in this way the infinite histories that violate the property, and then one constructs the product $\mathsf{N} \times \mathcal{A}_{\neg\psi}$ whose language is the language we are interested in, namely $Traces(\mathsf{N}) \cap (\mathcal{L}_\omega(\mathcal{A}_{\neg\psi}))$. Loosely speaking, the emptiness check on $\mathcal{L}_\omega(\mathsf{N} \times \mathcal{A}_{\neg\psi})$ reduces nicely to checking whether a final state of lies on a cycle in $\mathsf{N} \times \mathcal{A}_{\neg\psi}$. This translates to model checking the fixed LTL persistence property "*eventually forever* $\neg F$" where F is the disjunction of all final states of $\mathcal{A}_{\neg\psi}$. This property formalizes the requirement that no accepting state of $\mathcal{A}_{\neg\psi}$ will be visited infinitely often in a run, or, equivalently, that from a certain point on the state sequences induced by the histories of N never visit accepting states. Therefore:

$$\begin{aligned} \mathsf{N} \models \psi \quad &\text{iff} \quad \mathcal{L}_\omega(\mathsf{N} \times \mathcal{A}_{\neg\psi}) = \emptyset \\ &\text{iff} \quad \mathsf{N} \times \mathcal{A}_{\neg\psi} \models \Diamond\Box\neg F \end{aligned}$$

Checking the formula $\Diamond\Box\neg F$ amounts to checking whether no global transition leading to a marking containing a final state of the tester can be executed infinitely often. This is a *persistence checking problem*, and can be solved by means of algorithms that perform cycle detection, such as Nested DFS illustrated in [1].

Theorem 1. *Let* $\mathsf{N} = (N, \nu)$ *be a mcn-net with* $N = \langle S, T, F, m, \ell\rangle$ *and let* ψ *be an LTL formula over* $AP = S$*. Let* $\mathcal{A}_{\neg\psi} = \langle S', T', F', m', \ell'\rangle$ *be the state machine net encoding the Büchi automaton for* $\neg\psi$ *labelled over* $\Sigma = T$*. Then* $\mathsf{N} \models \psi$ *iff the product* $\mathsf{N} \times \mathcal{A}_{\neg\psi}$ *does not contain a reachable final state of* $\mathcal{A}_{\neg\psi}$ *that lies on a cycle.*

Notes about performance: The automata-based model checking approach we work with is PSPACE-hard, but there is room for optimizations in each step. In step (1), an LTL to Büchi automata naive translation produces very large automata which states can be reduced by using on-the-fly techniques as in [9], increasing performance of subsequent steps; in step (2) a more compact synchronization of the system and the automaton can be obtained by coupling only those transitions that trigger a change in the truth value of the states observed by the Büchi automaton and idling the others; this is the *stuttering synchronization* illustrated in [5] (it is however worth to note that this theory works nicely only on the fragment of LTL without the next operator - and consequently on the fragment of PLTL also without the previous modality). Many algorithms have been studied to solve the persistence check of step (3), [5] again studies a method that employs the unfolding procedure equipped with search strategies that intelligently exploits the concurrency of the stuttering synchronization.

5.2 Model checking reversible systems

We combine all the results seen so far to handle reversibility semantics in testing LTL formulae against arbitrary *mcn*-nets. Let $\mathsf{N} = (N, \nu)$ be an *mcn*-net and let $\mathsf{C} = ((C, \nu_C), \phi)$ be its unfolding, where $C = \langle B, E, F, c\rangle$ is an occurrence net, let $E' \subseteq E$ be a set of reversing transitions and let ψ be an LTL formula over $AP = B$:

1. Let $\overleftarrow{\mathsf{C}} = (\overleftarrow{C}, \nu_C)$ be the reversing net of N over E' as defined in Section 4,
2. Construct the Büchi automaton $\mathcal{A}_{\neg\psi}$ for the negation of ψ labeled by the events of $(\overleftarrow{C}, \nu_C)$ as hinted at Section 5.1,
3. Construct the product $\Sigma = \overleftarrow{\mathsf{C}} \times \mathcal{A}_{\neg\psi}$ as shown previously;
4. Test for illegal ω-traces in Σ, which is an infinite sequence violating the property (operatively a trace where accepting states of the NBA are visited infinitely often). If the test is positive, then there exists an infinite run of $\overleftarrow{\mathsf{C}}$ that violates ψ, and the model checker returns FALSE together with a sample run; otherwise it returns TRUE.

Example 7. Consider the Example 1, the unfolding is shown in Figure 3, and its enriching is partially depicted in Figure 5. We may try to prove that the thread A reads 1 then either she/he has never read 0 or if this is the case then the action of reading 0 has been undone. This property can be formalized in either PLTL (formula 1) or LTL (formula 2) as follows:

$$\Diamond\,(L1 \Rightarrow \boxdot\,\neg L0 \vee \Diamond\!\!\!\cdot(L0 \wedge \bigcirc\,\overline{L0})) \tag{1}$$

$$\Diamond\,L1 \Rightarrow (\Box\,\neg L0 \vee \Diamond\,(L0 \wedge \Diamond\,\overline{L0})) \tag{2}$$

where variables $L1$, $L0$ and $\overline{L0}$ represent the actions "*read 1*", "*read 0*" "*undo read 0*" respectively. The formula (1) reads as follows: " if the thread A has read 1 this implies that back in the past either he/she has never read 0 or he/she has read 0 and then afterwards that operation has been reversed". This formula can be re-written by just using forward modalities as (2). We instantiate these variables using the states of the unfolding of Figure 5, choosing $L1 = a_2'$, $L0 = a_2$ and $\overline{L0} = a_1$. These atomic propositions only concern thread A, its copy of the global variable x and the writing agent. We are able to verify the property which can be written using only *forward* operators.

6 Conclusions

In this paper, we have proposed a strategy to verify properties of reversible systems using model checking. This is, to the best of our knowledge, the first work to address this problem.

We model the system under verification as a product of sequential automata without internal concurrency. The unfolding of such a model captures all its possible behaviours. We then introduce reversibility at this level, following the

approach of [21]. Once we have enriched the unfolding with reversible transitions, we can apply again the unfolding, obtaining a *forward* only model which has the same behaviour (in terms of visited states) of the enriched one.

In model checking, properties are usually written in some logic. Linear temporal logic (LTL) is one of the most common logics, which enables writing formulae with future modalities. Past linear temporal logic (PLTL) extends LTL with past modalities, enabling reasoning about past. A well-known result [17] shows that PLTL and LTL have the same expressive power, so every PLTL formula can be rewritten as an equivalent LTL formula. Hence, classic LTL can be used to prove properties on reversible systems.

After reducing the model and the logic to only forward transitions, we have applied classical automata-based model checking consisting of constructing a Büchi automaton for the formula, and then building the product of the automaton with the system. We then have shown how this approach can be used to reason about reversibility with a shared memory example.

To the best of our knowledge, our approach is a first attempt to tackle the problem of model checking reversible systems. The closest work to ours is [33], where linear-time model checking techniques are applied to the verification of (closed) quantum systems. The idea is that a closed quantum system is inherently reversible, hence it is modelled by a class of reversible automaton called quantum automaton. We leave as future work a deeper comparison with [33].

Our work can be improved in several ways, for instance by implementing several optimisation algorithms to minimise the formula and the complexity of Büchi automaton. To reduce the complexity we could also resort to bounded model checking algorithms. Finally, we plan to build a fully-fledged tool with an usable interface wherein a user can model a reversible system and verify an arbitrary formula against the model.

Acknowledgments The authors thank the anonymous reviewers for their helpful comments that improved the quality of the paper.

References

1. C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.
2. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR 2004*, LNCS 3170. Springer, 2004.
3. V. Danos and J. Krivine. Transactions in RCCS. In *CONCUR 2005*, LNCS 3653. Springer, 2005.
4. J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6), 1991.
5. J. Esparza and K. Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008.
6. E. Fabre. Trellis processes : A compact representation for runs of concurrent systems. *Discret. Event Dyn. Syst.*, 17(3), 2007.
7. D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, LNCS 398. Springer, 1989.

8. L. Geatti, N. Gigante, A. Montanari, and G. Venturato. Past matters: Supporting ltl+past in the BLACK satisfiability checker. In *TIME 2021*, LIPIcs 206. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
9. R. Gerth, D. A. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV*, IFIP Conference Proceedings 38. Chapman & Hall, 1995.
10. E. Giachino, I. Lanese, and C. A. Mezzina. Causal-consistent reversible debugging. In *FASE 2014*, LNCS 8411. Springer, 2014.
11. U. Goltz and W. Reisig. The non-sequential behavior of Petri nets. *Information and Control*, 57(2/3), 1983.
12. V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of petri net unfoldings. *Acta Informatica*, 40(2), 2003.
13. I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. Concurrent flexible reversibility. In *ESOP 2013*, LNCS 7792. Springer, 2013.
14. I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversibility in the higher-order π-calculus. *Theor. Comput. Sci.*, 625, 2016.
15. I. Lanese and I. Phillips. Forward-reverse observational equivalences in CCSK. In *RC 2021*, LNCS 12805. Springer, 2021.
16. I. Lanese, U. P. Schultz, and I. Ulidowski. Reversible computing in debugging of erlang programs. *IT Prof.*, 24(1), 2022.
17. O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Logics of Programs*, LNCS 193. Springer, 1985.
18. K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV '92*, LNCS 663, 1993.
19. H. C. Melgratti, C. A. Mezzina, and G. M. Pinna. A distributed operational view of reversible prime event structures. In *LICS 2021*. IEEE, 2021.
20. H. C. Melgratti, C. A. Mezzina, and G. M. Pinna. Relating reversible petri nets and reversible event structures, categorically. In *FORTE 2023*, LNCS 13910. Springer, 2023.
21. H. C. Melgratti, C. A. Mezzina, and I. Ulidowski. Reversing place transition nets. *Log. Methods Comput. Sci.*, 16(4), 2020.
22. C. A. Mezzina, F. Tiezzi, and N. Yoshida. Rollback recovery in session-based programming. In *COORDINATION 2023*, volume 13908 of *LNCS*, pages 195–213. Springer, 2023.
23. A. Philippou and K. Psara. Reversible computation in nets with bonds. *J. Log. Algebraic Methods Program.*, 124:100718, 2022.
24. I. Phillips and I. Ulidowski. A hierarchy of reverse bisimulations on stable configuration structures. *Math. Struct. Comput. Sci.*, 22(2), 2012.
25. I. Phillips and I. Ulidowski. Event identifier logic. *Math. Struct. Comput. Sci.*, 24(2), 2014.
26. I. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *RC 2012*, LNCS 7581. Springer, 2012.
27. I. C. C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebraic Methods Program.*, 73(1-2), 2007.
28. G. M. Pinna. Reversing steps in membrane systems computations. In *CMC 2017*, LNCS 10725. Springer, 2017.
29. M. Schordan, T. Oppelstrup, D. R. Jefferson, and P. D. B. Jr. Generation of reversible C++ code for optimistic parallel discrete event simulation. *New Gener. Comput.*, 36(3):257–280, 2018.
30. I. Ulidowski, I. Phillips, and S. Yuen. Reversing event structures. *New Gener. Comput.*, 36(3), 2018.

31. M. Vassor and J.-B. Stefani. Checkpoint/rollback vs causally-consistent reversibility. In *RC 2018*, LNCS 11106. Springer, 2018.
32. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, LNCS 255. Springer, 1986.
33. M. Ying, Y. Li, N. Yu, and Y. Feng. Model-checking linear-time properties of quantum systems. *ACM Trans. Comput. Log.*, 15(3), 2014.