Testing Logical Diagrams in Power Plants: A Tale of LTL Model Checking

Aziz Sfar, David Carral, Dina Irofti, Madalina Croitoru

Testing Logical Diagrams in power plants: a tale of LTL model checking

Aziz Sfar[1,2][0000−0003−0359−0600], David Carral[1][0000−0001−7287−4709], Dina Irofti[2][0000−0002−9254−7980], and Madalina Croitoru[1][0000−0001−5456−7684]

[1] LIRMM, Inria, University of Montpellier, CNRS, Montpellier, France
[2] EDF R&D, France

Abstract. In this paper, we focus on the application of LTL (Linear Temporal Logic) model checking on logical diagrams (LD), which are a type of functional specification used for logical controllers in many nuclear power plants. The goal is to check properties on LDs and to generate counter examples serving as validation tests for logical controllers. We propose a sound and complete LTL encoding framework for LDs allowing the use of model checking (MC) and evaluate different MC techniques on real world LD to efficiently generate counterexamples for verifiable properties.

Keywords: Validation tests · Symbolic Model Checking · Linear Temporal Logic.

1 Introduction

We place ourselves in an applicative setting of the EDF company (French Electricity) where the behavior of Programmable Logical Controllers (PLC) used for nuclear power plants is periodically updated and tested manually within expert defined functional validation scenarios. A PLC is a system that embeds a program running control and protection functions. In a critical domain like nuclear energy production, the behavior of PLCs have to satisfy strict performance and safety requirements. Therefore, during the life-cycle of nuclear reactors, several updates are introduced to the PLC programs to keep up with last requirements defined by nuclear authorities. The updated behavior has to be validated by testing it against its functional specifications to ensure that no functional errors were generated following the modification process.

In the case of our application, the functional specifications of the behavior of PLCs are given in the form of Logical Diagrams (LD). These are graphical representations of a number of inputs and outputs connected through different types of logical blocks defining the expected behaviour of the machine (PLC). In a nuclear power plant, the number and size of LDs used as specifications for PLCs is enormous which makes the manual generation of test scenarios, as it is today, a tedious and time consuming task. Moreover, a Logical Diagram may contain feedback connections, i.e a number of blocks connected to each other in a loop, which could lead to cyclic behavior. This is a situation where one or many

outputs keep changing indefinitely without any change on the inputs making the specified behavior divergent.

In this paper, we address the problem of the stability property verification (i.e. absence of cyclic behaviour) on the Logical Diagrams and the generation of validation test cases for PLCs. Several works propose the transformation of the PLC description languages into formal representations like finite state models and timed automata to make use of the already existing tools and methods for test generation. For instance, Provost et al. [11] have proposed a translation of Grafcet specification models into Mealy machines to generate conformance test scenarios. In a previous work [12] inspired from [11], we have introduced a transformation of Logical Diagrams into state/transition graphs. The aim was to make use of the test generation techniques for transition systems as presented in [10] and [13] as well as the formal verification techniques to prove stability in Logical Diagrams. However, the exponential growth of the size of the generated transition systems makes their generation complicated for the biggest sized LDs. In [8], model-checking techniques were used to generate tests for PLC programs developed in Function Block Diagram (FBD). The paper describes the transformation of FBDs into timed automata and the use of a Model-Checker to generate test sequences. An experimental evaluation of their solution was made for many train control systems programs. However, to the authors knowledge, no similar approach was done for PLCs based on LD specifications used in nuclear power plants. In this paper, we propose to automatise the checking process of PLC behaviour specified in Logical Diagrams using tools from the Model Checking (*MC*) domain. We define a sound and complete transformation of Logical Diagrams into *Linear Temporal Logic (LTL)* formulas. We demonstrate how the stability property can be expressed into *LTL* expressions. We also show how test cases could be generated by formulating properties like the activation and deactivation of the outputs of the system in an LTL encoding and running Model-checkers on them. Finally, we empirically evaluate the efficiency of existing *MC* tools (NuSMV [5] and nuXmv [3]) on EDF logical diagrams. The empirical results obtained show that techniques like *k-MC* [6] is useful for checking the stability property while other techniques (*BDD* [7], *BMC* [1] and *SBMC* [9]) are timing out. The counterexample generation for activation and deactivation of outputs is possible for all techniques except *BDD*. We believe that the overview of the practical usability of *LTL* Model Checking tools for our real world scenario could be of use to researchers undergoing similar tasks (e.g. *LTL* encoding and *LTL* Model Checking for test generation).

This paper is organized as follows. In section 2 we set a formal definition of the Logical Diagram specification and we give an example. Section 3 introduces the *LTL* formalism and presents the *LTL* encoding of the logical diagram and the stability, activation and deactivation properties. Proof of the soundness and completeness of the transformation is given in section 4. Finally, section 5 discusses the application and evaluation of Model-Checking tools and techniques on our Logical Diagrams.

2 Logical Diagram

A *logical diagram* (LD) is a graphical representation of a set of interconnected blocks that perform logical functions. Before running a code implementing the logical diagram on a controller, a few properties of the logical diagram have to be studied. One of these properties is the stability property ensuring that the outputs of the logical diagram converge for some fixed inputs. Moreover, in order to test the implementation of a logical diagram on a controller, we would like to be able to generate input values, that allow to activate or deactivate a chosen output. Thus, we need to define the properties of activation and deactivation of outputs and use these properties in the verification and validation (V&V) process to validate the correct implementation of the logical diagram on the logical controller. In this section, we introduce a formal definition for logical diagrams and the properties of stability, activation and deactivation of outputs, and we illustrate them on the logical diagram in Figure 1.

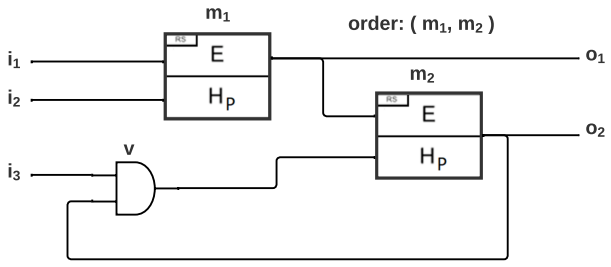

Fig. 1. A logical diagram example $LD = \langle V, E, \mathcal{O} \rangle$: $V(\mathsf{input}) = \{i_1, i_2, i_3\}$, $V(\mathsf{output}) = \{o_1, o_2\}$, $V(\mathsf{M}_E) = \{m_1, m_2\}$, $V(\mathsf{and}) = \{v\}$, $E = \{i_1 \xrightarrow{\mathrm{s}} m_1, i_2 \xrightarrow{\mathrm{r}} m_1, m_1 \xrightarrow{\mathrm{s}} m_2, v \xrightarrow{\mathrm{r}} m_2, i_3 \to v, m_2 \to v, m_1 \to o_1, m_2 \to o_2\}$, $\mathcal{O}(m_1) = 1$ and $\mathcal{O}(m_2) = 2$.

A logical diagram is formally defined by a set of *vertices* V, and a set of directed edges E connecting these vertices. Every vertex $v \in V$ has a type that defines the number and nature of its incoming and outgoing edges as formally defined in Def. 1. *Output* vertices are denoted by $V(\mathsf{output})$, *input* vertices by $V(\mathsf{input})$, *memory* vertices by $V(\mathsf{M})$ and finally vertices of the type '*and*', '*or*' and '*not*' are denoted $V(\mathsf{and})$, $V(\mathsf{or})$ and $V(\mathsf{not})$, respectively. In a logical diagram, we can find cycles. A cycle is a sequence of edges that starts and finishes at the same vertex v. In the diagrams used by EDF all cycles contain at least one memory vertex within the sequence. Consider the example of Figure 1: it contains three input vertices i_1, i_2 and i_3, two output vertices o_1 and o_2, two memory vertices m_1 and m_2, one vertex v of type 'and' and one cycle composed of v and m_2. Note that the memory vertices are represented in the diagram by set/reset (RS) blocks that have two inputs: E for set and H for reset. In the case of m_1, the incoming edge corresponding to the set is represented by $i_1 \xrightarrow{\mathrm{s}} m_1$

while $i_2 \xrightarrow{\mathrm{r}} m_1$ represents the reset edge. The incoming edges of the other vertices types is given by a simple arrow as in $i_3 \to v$. The letter 'p' in 'Hp' indicates that reset has priority over set. Then, we denote M_H as the type of the memory vertex. This is the case for both m_1 and m_2. Otherwise, the memory vertex is of type M_E with the priority expressed by 'Ep' in the set input of the block.

Definition 1 (Vertex type and LD). *The set of* vertex types *is* $\{\mathsf{input}, \mathsf{output}, \mathsf{and}, \mathsf{or}, \mathsf{not}, \mathsf{M}_E, \mathsf{M}_H\}$. *Henceforth, we associate every vertex* v *with a vertex type* $\mathsf{tp}(v)$. *Given a set* V *of vertices and a vertex type* t, *let* $V(t) = \{v \in V \mid \mathsf{tp}(v) = t\}$. *Moreover, let* $V(\mathsf{M}) = V(\mathsf{M}_E) \cup V(\mathsf{M}_H)$. *Moreover, a* directed edge *over* V *is an expression of the form* $u \to v$, $u \xrightarrow{s} v$, *or* $u \xrightarrow{r} v$ *with* $u, v \in V$.

A logical diagram *(LD) is a tuple* $\langle V, E, \mathcal{O}\rangle$ *such that* V *is a set of vertices,* E *is a set of directed edges (over* V*),* $\mathcal{O}$ *is a bijection from* $V(\mathsf{M})$ *to* $\{1, \ldots, |V(\mathsf{M})|\}$, *and all of the following hold:*

1. *If* $u \to v \in E$ *for some* $u, v \in V$*; then* $\mathsf{tp}(v)$ *is* output, and, or, *or* not.
2. *If* $u \xrightarrow{s} v$ *or* $u \xrightarrow{r} v$ *are in* E *for some* $u, v \in V$, *then* $\mathsf{tp}(v)$ *is* M_E *or* M_H.
3. *For every* $v \in V(\mathsf{not}) \cup V(\mathsf{output})$, *there is exactly one edge of the form* $u \to v \in E$.
4. *For every* $v \in V(\mathsf{and}) \cup V(\mathsf{or})$, *there are at least two edges of the form* $u \to v \in E$.
5. *For every* $v \in V(\mathsf{M})$, *there is exactly one edge of the form* $u \xrightarrow{s} v \in E$ *and exactly one edge of the form* $w \xrightarrow{r} v \in E$.
6. *For every* $v_1, \ldots, v_n \in V$, *either* $v_n \to v_1 \notin E$ *or* $v_{i-1} \to v_i \notin E$ *for some* $2 \leq i \leq n$. *That is, every cycle have to traverse at least one memory vertex.*

Each vertex in a logical diagram has an associated logical value called the output of the vertex. It takes the value *true* referred to by $\top$ or false referred to by $\bot$. Furthermore, the vertex output is evaluated in accordance to increasing time steps and can vary from a step to another. The time step k starts at zero. Initially, i.e. at $k = 0$, output values of memory and input vertices are given by an *initializing function* f. Then, the output value of every vertex v at a time step $k \geq 0$, is defined by the function $LD_f(v, k)$. Specifically, the output of memory vertices are evaluated one after the other according to an evaluation order specified by the function $\mathcal{O}$. The output value of only one memory vertex can change at each time step if it is its turn, while the others maintain the same value from the previous step. Note that the output of a memory vertex of type M_E (resp. M_H) is evaluated to True, denoted $\top$, (resp. False, denoted $\bot$), when both set and reset inputs are evaluated to True. The complete output evaluation for different vertices types is given by Def. 3.

Definition 2 (ev function). *The set* $\mathcal{B}$ *of Boolean expressions is defined by the grammar* $\mathcal{B} ::= \top \mid \bot \mid \neg\mathcal{B} \mid \mathcal{B} \wedge \mathcal{B} \mid \mathcal{B} \vee \mathcal{B}$. *We define* $\mathsf{ev} : \mathcal{B} \mapsto \{\top, \bot\}$ *such that for every* $a, b \in \mathcal{B}$*:*

- $\mathsf{ev}(\top) = \top$.
- $\mathsf{ev}(\bot) = \bot$.

- $\mathsf{ev}(\neg a) = \top$ *iff* $\mathsf{ev}(a) = \bot$.
- $\mathsf{ev}(a \vee b) = \top$ *iff* $\mathsf{ev}(a) = \top$ *or* $\mathsf{ev}(b) = \top$.
- $\mathsf{ev}(a \wedge b) = \top$ *iff* $\mathsf{ev}(a) = \top$ *and* $\mathsf{ev}(b) = \top$.

Definition 3 (Output function). *Consider a diagram* $LD = \langle V, E, \mathcal{O}\rangle$ *and an* initializing function f *for* LD*; that is, a total function from* $V(\mathsf{input}) \cup V(\mathsf{M})$ *to* $\{\top, \bot\}$. *For every vertex* $v \in V(\mathsf{M})$, *let* $\mathsf{turn}_{\mathcal{O}}(v)$ *be the set that contains* $\mathcal{O}(v) + n \cdot |V(\mathsf{M})|$ *for every* $n \geq 0$. *Also, for every* $v \in V$ *and* $k \geq 0$, *we define* $LD_f(v, k) \in \{\top, \bot\}$ *as follows:*

- *If* $v \in V(\mathsf{input})$, *then* $LD_f(v, k) = f(v)$.
- *If* $v \in V(\mathsf{output})$, *then* $LD_f(v, k) = \mathsf{ev}(LD_f(u, k))$ *where* u *is the vertex in* V *with* $u \to v \in E$.
- *If* $v \in V(\mathsf{not})$, *then* $LD_f(v, k) = \mathsf{ev}(\neg LD_f(u, k))$ *where* u *is the vertex with* $u \to v \in E$.
- *If* $v \in V(\mathsf{or})$, *then* $LD_f(v, k) = \mathsf{ev}(\bigvee_{u \in P} LD_f(u, k))$ *where* $P = \{u \in V \mid u \to v \in E\}$.
- *If* $v \in V(\mathsf{and})$, *then* $LD_f(v, k) = \mathsf{ev}(\bigwedge_{u \in P} LD_f(u, k))$ *where* $P = \{u \in V \mid u \to v \in E\}$.
- *If* $v \in V(\mathsf{M})$ *and* $k = 0$, *then* $LD_f(v, k) = f(v)$.
- *If* $v \in V(\mathsf{M})$, $k \geq 1$, *and* $k \notin \mathsf{turn}_{\mathcal{O}}(v)$ *then* $LD_f(v, k) = LD_f(v, k-1)$.
- *If* $v \in V(\mathsf{M})$ *and* $k \in \mathsf{turn}_{\mathcal{O}}(v)$, *we consider two cases:*
  - *If* $\mathsf{tp}(v) = M_E$, *then* $LD_f(v, k) = \mathsf{ev}((LD_f(v, k-1) \wedge \neg LD_f(h, k-1)) \vee LD_f(e, k-1))$.
  - *If* $\mathsf{tp}(v) = M_H$, *then* $LD_f(v, k) = \mathsf{ev}(\neg LD_f(h, k-1) \wedge (LD_f(v, k-1) \vee LD_f(e, k-1)))$.

  In the above, $h \in V$ *with* $h \xrightarrow{r} v \in E$ *and* $e \in V$ *with* $e \xrightarrow{s} v \in E$.

In the example of Figure 1, the evaluation order of LD is $\mathcal{O}(m_1) = 1$ and $\mathcal{O}(m_2) = 2$. The output function LD_f depends on this evaluation order as well as the initializing function as established in Def. 3. Note that input vertices maintain the value given to them in the initial step.

Thus far, we have defined the different elements composing a logical diagram and the output evaluation function. We define in the sequel the properties of stability, activation and deactivation of outputs. For the stability property, we want to make sure that in some future time step k, the output values of vertices converge. In other words, having an initializing function f, the logical output value of every vertex should not oscillate indefinitely between the values true and false, from a step to the next. For this to be avoided, it suffices that the outputs of memory vertices converge as mentioned in Def. 4. In this case, we say that the logical diagram is stable for the initializing function f. We also say that a logical diagram is uniformly stable if it is stable for every possible input scenario, i.e. for every initializing function f.

Definition 4 (stability). *A logical diagram* $LD = \langle V, E, \mathcal{O}\rangle$ *is* stable *for some initializing function* f, *if there exists* $k' \geq 0$ *such that for every* $k \geq k'$ *and* $v \in V(\mathsf{M})$, *we have* $LD_f(v, k) = LD_f(v, k+1)$. *We say that a logical Diagram* LD *is* uniformly stable *if it is stable for every initializing function* f.

To illustrate, we consider the initializing function $f(i_1) = \top$, $f(i_2) = \bot$, $f(i_3) = \top$, $f(m_1) = \bot$ and $f(m_2) = \bot$ for the logical diagram of Figure 1. By applying the evaluation function on the memory vertex m_2, we can see that $LD_f(m_2, k)$ is not equal to $LD_f(m_2, k+2)$ for every $k \geq 0$. Therefore, LD is not stable for f. Hence LD is not uniformly stable.

Once the stability is verified, the next step consists in generating initializing functions that allow to set to true (i.e activate) or set to false (i.e deactivate) an output vertex o. It is also possible to set some initial conditions over the input and output values before generating scenarios. We look to answer questions like: is there an initializing function f that eventually activates an output o, when initially, o is deactivated (i.e. $LD_f(o, 0) = \bot$) and the input i is activated (i.e. $LD_f(i, 0) = \top$). The chosen initial conditions are given by an initial configuration function $\mathsf{conf}_{\mathcal{S}}$ setting the initial values of a set of vertices $\mathcal{S} \subseteq V(\mathsf{output}) \cup V(\mathsf{input})$. Then, we say that the output o is activated (resp. deactivated) for an initializing function f, if f satisfies the initial configuration $\mathsf{conf}_{\mathcal{S}}$, and grants the activation (resp. deactivation) of o in some future step k (as per Def. 5).

Definition 5. *Consider a logical diagram $LD = \langle V, E, \mathcal{O} \rangle$ and an* initial configuration *function $\mathsf{conf}_{\mathcal{S}}$; that is a function from $\mathcal{S} \subseteq V(\mathsf{input}) \cup V(\mathsf{output})$ to $\{\top, \bot\}$. An output vertex $o \in V(\mathsf{output})$ is* activated *(resp.* deactivated *) for some initial configuration $\mathsf{conf}_{\mathcal{S}}$, if there is an initializing function f and $k' \geq 0$ such that; for every $v \in \mathcal{S}$, $LD_f(v, 0) = \mathsf{conf}_{\mathcal{S}}(v)$ and for every $k \geq k'$, $LD_f(o, k) = \top$ (resp. $LD_f(o, k) = \bot$).*

Consider for instance the logical diagram of Figure 1, the set $\mathcal{S} = \{o_1, o_2\}$ and the initial configuration $\mathsf{conf}_{\mathcal{S}}(o_1) = \bot$ and $\mathsf{conf}_{\mathcal{S}}(o_2) = \bot$. Let f_1 be an initializing function such that $f_1(i_1) = \top$, $f_1(i_2) = \bot$, $f_1(i_3) = \bot$, $f_1(m_1) = \bot$ and $f_1(m_2) = \bot$. By Def. 3, we have $LD_f(o_1, 0) = \bot$, $LD_f(o_2, 0) = \bot$ and $LD_f(o_2, k) = \top$ for every $k \geq 2$. Therefore, o_2 is activated for f_1.

3 LTL encoding of logical diagrams

In this section, we propose an encoding framework of the logical diagrams and properties into a set of *Linear Temporal Logic* (*LTL*) formulas. An *LTL* formula is built up from a set of *propositional variables* AP and a set of logical and temporal operators. The set $\mathcal{L}$ of *LTL* formulas is defined by the following grammar: $\mathcal{L} ::= p \mid \top \mid \bot \mid \neg\mathcal{L} \mid \mathcal{L} \wedge \mathcal{L} \mid \mathcal{L} \vee \mathcal{L} \mid \mathcal{L} \rightarrow \mathcal{L} \mid \mathcal{L} \leftrightarrow \mathcal{L} \mid \bigcirc\mathcal{L} \mid \Box\mathcal{L} \mid \Diamond\mathcal{L}$ where $p \in AP$. A *word* $\mathcal{W}$ is a sequence $s_0, s_1, \ldots$ of subsets of AP. Each element s_i of the sequence is referred to by $\langle \mathcal{W}, i \rangle$. Given a word $\mathcal{W}$, a propositional variable p, two *LTL* formulas φ and ψ, and some $i \geq 0$; we write:

- $\langle \mathcal{W}, i \rangle \models p$ iff $p \in s_i$.
- $\langle \mathcal{W}, i \rangle \models \top$.
- $\langle \mathcal{W}, i \rangle \not\models \bot$.
- $\langle \mathcal{W}, i \rangle \models \neg\varphi$ iff $\langle \mathcal{W}, i \rangle \not\models \varphi$.
- $\langle \mathcal{W}, i \rangle \models \varphi \wedge \psi$ iff $\langle \mathcal{W}, i \rangle \models \varphi$ and $\langle \mathcal{W}, i \rangle \models \psi$.

- $\langle \mathcal{W}, i\rangle \models \varphi \vee \psi$ iff $\langle \mathcal{W}, i\rangle \models \varphi$ or $\langle \mathcal{W}, i\rangle \models \psi$.
- $\langle \mathcal{W}, i\rangle \models \varphi \rightarrow \psi$ iff $\langle \mathcal{W}, i\rangle \models \neg\varphi \vee \psi$.
- $\langle \mathcal{W}, i\rangle \models \varphi \leftrightarrow \psi$ iff $\langle \mathcal{W}, i\rangle \models (\psi \rightarrow \varphi) \wedge (\varphi \rightarrow \psi)$.
- $\langle \mathcal{W}, i\rangle \models \bigcirc\varphi$ iff $\langle M, i+1\rangle \models \varphi$.
- $\langle \mathcal{W}, i\rangle \models \Diamond\varphi$ iff $\langle M, j\rangle \models \varphi$ for some $j \geq i$.
- $\langle \mathcal{W}, i\rangle \models \Box\varphi$ iff $\langle M, j\rangle \models \varphi$ for every $j \geq i$.

We say that a word $\mathcal{W}$ satisfies an *LTL* formula Φ at a time step i and we write $\langle M, i\rangle \models \Phi$. Moreover, $\mathcal{W}$ is a *model* of Φ ($\mathcal{W} \models \Phi$) if $\mathcal{W}$ satisfies Φ at the time step 0. An *LTL* theory is a defined finite set of *LTL* formulas. A word $\mathcal{W}$ is a model of an *LTL* theory $\mathcal{T}$ if it is a model of every formula $\Phi \in \mathcal{T}$. An *LTL* theory $\mathcal{T}_1$ entails another theory $\mathcal{T}_2$ ($\mathcal{T}_1 \models \mathcal{T}_2$) when every model of $\mathcal{T}_1$ is also a model of $\mathcal{T}_2$.

3.1 LTL encoding of logical diagrams and initializing functions

We introduce a sound and complete *LTL* encoding of logical diagrams. The encoding will be given in two *LTL* theories. The first, denoted $\mathcal{T}_{LD}$, encodes the vertices, edges and characteristics of a logical diagram *LD*; the second $\mathcal{T}_f$ encodes the initial output values given by an initializing function f.

Let $LD = \langle V, E, \mathcal{O}\rangle$ be a logical diagram. We define the *LTL* vocabulary AP_{LD} of propositional variables: we associate to every vertex $v \in V$ a propositional variable t_v and to each number $i \geq 1$ a variable t_i. The set AP_{LD} is composed of these t_v and t_i. Then, the *LTL* theory $\mathcal{T}_{LD}$ contains an *LTL* formula for every vertex $v \in V$. This formula represents the output function of v at every time step. On the logical diagram of Figure 1, the *LTL* theory $\mathcal{T}_{LD}$ of our example contains the formulas $\{\Box(t_{o_1} \leftrightarrow t_{m_1}), \Box(t_{o_2} \leftrightarrow t_{m_2})\}$ for the vertices o_1 and o_2, the formula $\Box(t_v \leftrightarrow t_{i_3} \wedge t_{m_2})$ for the vertex v, and $\{\Box(\bigcirc t_{m_1} \leftrightarrow ((\neg t_1 \wedge t_{m_1}) \vee (t_1 \wedge \neg t_{i_2} \wedge (t_{m_1} \vee t_{i_1})))),$
$\Box(\bigcirc t_{m_2} \leftrightarrow ((\neg t_2 \wedge t_{m_2}) \vee (t_2 \wedge \neg t_v \wedge (t_{m_2} \vee t_{m_1}))))\}$ for the vertices m_1 and m_2. In the last two formulas, the variable t_1 indicates the evaluation turn of m_1, and likewise with t_2 and m_2. Therefore, we add to $\mathcal{T}_{LD}$ the formulas that translate the ordered evaluation of memory vertices. We construct these formulas in a way that ensures that $t_{\mathcal{O}(v)}$ is exclusively true at the time steps dedicated for the evaluation of the memory vertex v according to $\mathcal{O}(v)$. For the logical diagram *LD* in Fig. 1, we add to $\mathcal{T}_{LD}$ the set of *LTL* formulas : $\{(t_1 \wedge \neg t_2), \Box(t_1 \leftrightarrow \bigcirc(\neg t_1 \wedge t_2)), \Box(t_2 \leftrightarrow \bigcirc(\neg t_2 \wedge t_1))\}$. Finally, we add to $\mathcal{T}_{LD}$ the set of formulas that translate the fact that input vertices of the logical diagram keep the same values assigned to them in the initial time step. We do this by adding the *LTL* formula $(\Box t_v \vee \Box\neg t_v)$ for every vertex $v \in V(\mathsf{input})$. The formulas $\{(\Box t_{i_1} \vee \Box\neg t_{i_1}), (\Box t_{i_2} \vee \Box\neg t_{i_2}), (\Box t_{i_3} \vee \Box\neg t_{i_3})\}$ belong therefore to the theory $\mathcal{T}_{LD}$ of previous example. The full definition of $\mathcal{T}_{LD}$ is established in Def. 6

Definition 6 ($\mathcal{T}_{LD}$). *The set of propositional variables AP_{LD} contains a variable t_v for every $v \in V$ and a variable t_i for every $i \geq 1$. Consider a logical*

diagram $LD = \langle V, E, \mathcal{O} \rangle$ *and some initializing function* f*. We define the* $\mathcal{T}_{LD}$*, the LTL theory over* AP_{LD} *that contains all the following formulas :*

- $(t_1 \wedge \bigwedge_{i \in \{2,\ldots,|V(\mathsf{M})|\}} \neg t_i)$.
- $\bigwedge_{i \in \{1,\ldots,|V(\mathsf{M})|-1\}} \Box(t_i \leftrightarrow \bigcirc(\neg t_i \wedge t_{i+1}))$.
- $\Box(t_{|V(\mathsf{M})|} \leftrightarrow \bigcirc(\neg t_{|V(\mathsf{M})|} \wedge t_1))$).
- *For every* $v \in V(\mathsf{input})$*, add* $(\Box t_v \vee \Box \neg t_v)$.
- *For every* $v \in V(\mathsf{output})$*, add* $\Box(t_u \leftrightarrow t_v)$ *where* $u \in V$ *with* $u \rightarrow v \in E$.
- *For every* $v \in V(\mathsf{not})$*, add* $\Box(t_u \leftrightarrow \neg t_v)$ *where* $u \in V$ *with* $u \rightarrow v \in E$.
- *For every* $v \in V(\mathsf{or})$*, add* $\Box(t_v \leftrightarrow \bigvee_{u \in P} t_u)$ *where* $P = \{u \mid u \rightarrow v \in E\}$.
- *For every* $v \in V(\mathsf{and})$*, add* $\Box(t_v \leftrightarrow \bigwedge_{u \in P} t_u)$ *where* $P = \{u \mid u \rightarrow v \in E\}$.
- *if* $\mathsf{tp}(v) = M_E$*, then add*

$$\Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge ((t_v \wedge \neg t_h) \vee t_e))\big)\Big)$$

- *if* $\mathsf{tp}(v) = M_H$ *with* $i = \mathcal{O}(v)$*, then:*

$$\Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge \neg t_h \wedge (t_v \vee t_e))\big)\Big)$$

In the above, h *is the vertex with* $h \xrightarrow{r} v \in E$ *and* e *is the vertex with* $e \xrightarrow{s} v \in E$.

Let us now consider some initializing function f. The second *LTL* encoding is given by the *LTL* theory $\mathcal{T}_f$ that contains, for every vertex $v \in V(\mathsf{input}) \cup V(\mathsf{M})$, the *LTL* formula $(t_v \leftrightarrow f(v))$.

Definition 7 ($\mathcal{T}_f$). *Consider some initiliazing function* f *for a logical diagram* $LD = \langle V, E, \mathcal{O} \rangle$*. Then, let* $\mathcal{T}_f$ *be the LTL theory over* AP_{LD} *such that* $(t_v \leftrightarrow f(v)) \in \mathcal{T}_f$ *for every* $v \in V(\mathsf{input}) \cup V(\mathsf{M})$.

Consider a word $\mathcal{W}$ that is a model of the *LTL* encoding $\mathcal{T}_{LD}$ of a logical diagram LD and $\mathcal{T}_f$ of an initializing function f. At each time step k, $\mathcal{W}$ satisfies the variables t_v of the vertices v whose outputs $LD_f(v, k)$ are evaluated to True. In other words, the proposed *LTL* encoding is sound and complete as stated by Theorem 1 proven in section 4.

Theorem 1. *Consider a logical diagram* $LD = \langle V, E, \mathcal{O} \rangle$*, some initializing function* f*, and the LTL theories* $\mathcal{T}_{LD}$ *and* $\mathcal{T}_f$*. Let* $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_f$*. Then for every* $v \in V$ *and* $k \geq 0$*,* $LD_f(v, k) = \top \Leftrightarrow \langle \mathcal{W}, k \rangle \models t_v$*, where* $t_v \in AP_{LD}$.

3.2 LTL encoding of properties

In section 2, three types of properties were defined for logical diagrams. Here, we introduce an *LTL* theory for each different property. We first specify the *LTL* theory $\mathcal{T}_{stable}$ for the stability property. Then, we define the *LTL* theories $\mathcal{T}^o_{act}$ and $\mathcal{T}^o_{deact}$, respectively for the activation and deactivation properties.

Let us consider a logical diagram $LD = \langle V, E, \mathcal{O} \rangle$ and an initializing function f. The stability of LD is satisfied if in some future the output values of all

the memory vertices remain unchanged. This can easily be expressed using the temporal operators offered by *LTL*. We encode the stability property of a logical diagram in the *LTL* theory $\mathcal{T}_{stable}$ that contains the formula $\Diamond(\Box t_v \vee \Box\neg t_v)$ for every $v \in V(\mathsf{M})$. Using the *LTL* encoding, we can establish whether or not a logical diagram is stable. In fact, if the model $\mathcal{W}$ of the theories $\mathcal{T}_{LD}$ and $\mathcal{T}_f$ is also a model of $\mathcal{T}_{stable}$, then the logical diagram *LD* is stable for f as stated in Theorem 2. In the case of the logical diagram of Figure 1, let f_1 be an initializing function such that $f_1(i_1) = \top$, $f_1(i_2) = \bot$, $f_1(i_3) = \top$, $f_1(m_1) = \bot$ and $f_1(m_2) = \bot$. Then, $\mathcal{T}_{f_1} = \{t_{i_1} \leftrightarrow \top, t_{i_2} \leftrightarrow \bot, t_{i_3} \leftrightarrow \top, t_{m_1} \leftrightarrow \bot, t_{m_2} \leftrightarrow \bot\}$. The *LTL* encoding of the stability property is $\mathcal{T}_{stable} = \{\Diamond(\Box t_{m_1} \vee \Box\neg t_{m_1}), \Diamond(\Box t_{m_2} \vee \Box\neg t_{m_2})\}$. The *LTL* model checking proves that the satisfaction relation $\mathcal{T}_{LD} \cup \mathcal{T}_{f_1} \models \mathcal{T}_{stable}$ is false. Therefore, we conclude that the logical diagram is not stable for the initializing function f_1. This also means that *LD* is not uniformly stable.

Theorem 2. *Consider a logical diagram $LD = \langle V, E, \mathcal{O}\rangle$, some initializing function f, and the LTL theories $\mathcal{T}_{LD}$, $\mathcal{T}_f$ and $\mathcal{T}_{stable}$. Then, LD is stable for f if $\mathcal{T}_{LD} \cup \mathcal{T}_f \models \mathcal{T}_{stable}$. The logical diagram is uniformly stable if $\mathcal{T}_{LD} \models \mathcal{T}_{stable}$.*

We next encode the activation and deactivation properties. Let $o \in V(\mathsf{output})$ be an output vertex and $\mathcal{S}$ a subset of vertices. The *LTL* theory $\mathcal{T}^o_{act}$ represents the activation property of o for an initial configuration $\mathsf{conf}_\mathcal{S}$. For every vertex $v \in \mathcal{S}$, it contains the formula $(t_v \leftrightarrow \mathsf{conf}_\mathcal{S}(v))$ which translates the initial configuration. It also contains the formula $(\Diamond\Box t_o)$ that translates the activation of the output o in some future. Likewise, the *LTL* theory $\mathcal{T}^o_{deact}$ that encodes the deactivation property of the output vertex o for the initial configuration $\mathsf{conf}_\mathcal{S}$, contains the formulas $(\Diamond\Box\neg t_o)$ and $(v \leftrightarrow \mathsf{conf}_\mathcal{S}(v))$, for every $v \in \mathcal{S}$. Theorem 3 states that the satisfaction of the *LTL* theory $\mathcal{T}_{LD}$ and the theories of the activation and deactivation properties means that the encoded logical diagram also satisfies these properties. For the illustration example of Figure 1, consider the subset of vertices $\mathcal{S} = \{o_1, o_2\}$ and the initial configuration $\mathsf{conf}_\mathcal{S}(o_1) = \bot$ and $\mathsf{conf}_\mathcal{S}(o_2) = \bot$. The activation property encoding for the output vertex o_2 is $\mathcal{T}^{o_2}_{act} = \{(t_{o_1} \leftrightarrow \bot), (t_{o_2} \leftrightarrow \bot), \Diamond\Box t_{o_2}\}$. We are looking to find some initializing function f_1 such that $\mathcal{T}_{LD} \cup \mathcal{T}_{f_1} \models \mathcal{T}^{o_2}_{act}$. In order to do so, we run an *LTL* model checking on the formula $\mathcal{T}_{LD} \cup \mathcal{T}^{o_2}_{act} \models \bot$. Using a *BDD* based model checking algorithm [7], the property is declared as false with the counterexample $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_{f_1} \cup \mathcal{T}^{o_2}_{act}$ where $f_1(i_1) = \top$, $f_1(i_2) = \bot$, $f_1(i_3) = \bot$, $f_1(m_1) = \bot$ and $f_1(m_2) = \bot$. The output o_2 is therefore activated for f_1.

Theorem 3. *Consider a logical diagram $LD = \langle V, E, \mathcal{O}\rangle$ and the LTL theory $\mathcal{T}_{LD}$. An output vertex $o \in V(\mathsf{output})$ is activated (resp. deactivated) for some initial configuration $\mathsf{conf}_\mathcal{S}$, if there is an initializing function f such that $\mathcal{T}_{LD} \cup \mathcal{T}_f \models \mathcal{T}^o_{act}$ (resp. $\mathcal{T}_{LD} \cup \mathcal{T}_f \models \mathcal{T}^o_{deact}$).*

4 Proofs

In this section we include the proof of Theorem 1. Lines will be enumerated for readability purposes and to make it easier to refer to different elements of the

proof. The proof of Theorem 1 is established on three steps (A), (B) and (C). Moreover, we introduce and prove the following auxiliary lemma, which is later used for the proof of (C).

Lemma 1. *Consider a logical diagram $LD = \langle V, E, \mathcal{O}\rangle$, and the LTL theory $\mathcal{T}_{LD}$. Let $\mathcal{W} \models \mathcal{T}_{LD}$ and $v \in V(\mathsf{M})$ then, for every $k \geq 1$ $\langle \mathcal{W}, k-1\rangle \models t_{\mathcal{O}(v)}$ iff $k \in \mathsf{turn}_{\mathcal{O}}(v)$.*

Proof. Proof of Lemma 1.

1. Let $LD = \langle V, E, \mathcal{O}\rangle$ be a logical diagram and $\mathcal{T}_{LD}$ be the corresponding LTL theory over AP_{LD}.
2. Let $\mathcal{W}$ be a word such that $\mathcal{W} \models \mathcal{T}_{LD}$.
3. We prove the following by induction: if $k \in \mathsf{turn}_{\mathcal{O}}(v)$ then $\langle \mathcal{W}, k-1\rangle \models t_{\mathcal{O}(v)}$ for every $k \geq 1$.
4. We prove the statement for the base case: k=1.
   4.a By Def. 3, we have $k = 1 \in \mathsf{turn}_{\mathcal{O}}(v)$ where $v \in V(\mathsf{M})$ and $\mathcal{O}(v) = 1$.
   4.b By Def 6, we have $\langle \mathcal{W}, 0\rangle \models t_1$.
   4.c Based on 4.a and 4.b, the statement 3 holds for $k = 1$.
5. We assume that the statement in 3 holds for some $k \geq 1$. We then prove that it also holds for $k + 1$ and $v \in V(\mathsf{M})$.
6. Let $k \geq 1$ where $k + 1 \in \mathsf{turn}_{\mathcal{O}}(v)$. We prove that $\langle \mathcal{W}, k\rangle \models t_{\mathcal{O}(v)}$.
7. If $\mathcal{O}(v) \in \{2 \ldots |V(\mathsf{M})|\}$:
   7.a As established in 6, $k+1 \in \mathsf{turn}_{\mathcal{O}}(v)$. Then by Def 3, $k \in \mathsf{turn}_{\mathcal{O}}(w)$ where $w \in V(\mathsf{M})$ and $\mathcal{O}(w) = \mathcal{O}(v) - 1$.
   7.b By 5 and 7.a, we conclude that $\langle \mathcal{W}, k-1\rangle \models t_{\mathcal{O}(v)-1}$.
   7.c By Def. 6 we have $\mathcal{W} \models \bigwedge_{i\in\{1,\ldots,|V(\mathsf{M})|-1\}} \Box(t_i \leftrightarrow \bigcirc(\neg t_i \wedge t_{i+1}))$.
   7.d Based on 7.c, $\langle \mathcal{W}, k-1\rangle \models (t_{\mathcal{O}(v)-1} \leftrightarrow \bigcirc(\neg t_{\mathcal{O}(v)-1} \wedge t_{\mathcal{O}(v)}))$.
   7.e From 7.b and 7.d we conclude that $\langle \mathcal{W}, k\rangle \models t_{\mathcal{O}(v)}$. Thus, the statement 3 is true for $k + 1$.
8. Like in 7, we use 5, Def 3 and Def 6 to prove that 3 also holds when $\mathcal{O}(v) = 1$.
9. Based on 7.e and 8, the statement 3 holds for $k + 1$. Consequently, for every $k \geq 1$, if $k \in \mathsf{turn}_{\mathcal{O}}(v)$ then $\langle \mathcal{W}, k-1\rangle \models t_{\mathcal{O}(v)}$.
10. **Next**, we prove the following by induction: for $k \geq 1$, if $\langle \mathcal{W}, k-1\rangle \models t_{\mathcal{O}(v)}$ then $k \in \mathsf{turn}_{\mathcal{O}}(v)$.
11. We prove the statement 10 for the base case: $k = 1$.
    11.a By Def. 6 we have $\mathcal{W} \models (t_1 \wedge \bigwedge_{i\in\{2,\ldots,|V(\mathsf{M})|\}} \neg t_i)$. Thus, $\langle \mathcal{W}, 0\rangle \models t_1$.
    11.b By Def. 3, we have $k = 1 \in \mathsf{turn}_{\mathcal{O}}(v)$ where $v \in V(\mathsf{M})$ and $\mathcal{O}(v) = 1$.
    11.c Based on 11.a and 11.b, the statement 10 holds for $k = 1$.
12. We now assume that the statement 10 holds for some $k \geq 1$. We prove that it also holds for $k + 1$.
13. Let $v \in V(\mathsf{M})$. Assuming that $\langle M, k\rangle \models t_{\mathcal{O}(v)}$, we prove that $k+1 \in \mathsf{turn}_{\mathcal{O}}(v)$.
14. If $\mathcal{O}(v) \in \{2 \ldots |V(\mathsf{M})| - 1\}$:
    14.a By Def. 6 we have $\mathcal{W} \models \bigwedge_{i\in\{1,\ldots,|V(\mathsf{M})|-1\}} \Box(t_i \leftrightarrow \bigcirc(\neg t_i \wedge t_{i+1}))$.
    14.b Based on 14.a we have: $\langle \mathcal{W}, k\rangle \models (t_{\mathcal{O}(v)} \leftrightarrow \bigcirc(\neg t_{\mathcal{O}(v)} \wedge t_{\mathcal{O}(v)+1}))$.
    14.c By 13 we know that $\langle M, k\rangle \models t_{\mathcal{O}(v)}$. Therefore, based on 14.b we have $\langle M, k+1\rangle \models t_{\mathcal{O}(v)+1}$ and $\langle M, k+1\rangle \not\models t_{\mathcal{O}(v)}$.

14.d Based on 14.a, we have $\langle \mathcal{W}, k \rangle \models (t_{\mathcal{O}(v)-1} \leftrightarrow \bigcirc(\neg t_{\mathcal{O}(v)-1} \wedge t_{\mathcal{O}(v)}))$
14.e From 14.c we have $\langle M, k+1 \rangle \not\models t_{\mathcal{O}(v)}$. Thus, based on 14.d we conclude that $\langle M, k \rangle \not\models t_{\mathcal{O}(v)-1}$.
14.f Based on 14.a, we have $\langle \mathcal{W}, k-1 \rangle \models (t_{\mathcal{O}(v)-1} \leftrightarrow \bigcirc(\neg t_{\mathcal{O}(v)-1} \wedge t_{\mathcal{O}(v)}))$.
14.g From 13 and 14.e we have $\langle M, k \rangle \models t_{\mathcal{O}(v)}$ and $\langle M, k \rangle \not\models t_{\mathcal{O}(v)-1}$. Thus, by 14.f $\langle M, k-1 \rangle \models t_{\mathcal{O}(v)-1}$.
14.h As per 12, the statement 10 holds for k. By 14.g, $\langle M, k-1 \rangle \models t_{\mathcal{O}(v)-1}$. Thus, $k \in \mathsf{turn}_{\mathcal{O}}(w)$ where $w \in V(\mathsf{M})$ and $\mathcal{O}(w) = \mathcal{O}(v) - 1$.
14.i Based on 14.h and Def. 3 we conclude that $k+1 \in \mathsf{turn}_{\mathcal{O}}(v)$. The statement 10 is therefore true for $k+1$ as per 13.

15. Like in 14, we use 12 and Def 6 to prove that statement 10 also holds when $\mathcal{O}(v) = 1$ and $\mathcal{O}(v) = |V(\mathsf{M})|$.
16. Based on 14.i and 15, the statement 10 holds for $k+1$. Consequently, for every $k \geq 1$, if $\langle \mathcal{W}, k-1 \rangle \models t_{\mathcal{O}(v)}$ then $k \in \mathsf{turn}_{\mathcal{O}}(v)$.
17. **Conclusion:** By 9 and 16 we conclude that Lemma 1 is true.

Next, we give the proof of Theorem 1. Consider a logical diagram $LD = \langle V, E, \mathcal{O} \rangle$, some initializing function f, and a word $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_f$; we prove the following: $LD_f(v, k) = \top \Leftrightarrow \langle \mathcal{W}, k \rangle \models t_v$ where $t_v \in AP_{LD}$

(A) for every $v \in V(\mathsf{input})$ and $k \geq 0$,
(B) for every $v \in V$ and $k = 0$,
(C) for every $v \in V$ and $k \geq 1$.

Proof. Proof of Theorem 1 part (A): $v \in V(\mathsf{input})$ and $k \geq 0$.

18. Consider a logical diagram $LD = \langle V, E, \mathcal{O} \rangle$ and an initializing function f. The *LTL* theories are $\mathcal{T}_{LD}$ and $\mathcal{T}_f$. Let $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_f$. Let $v \in V(\mathsf{input})$.
19. By Def. 3 we have $LD_f(v, k) = \top$ iff $f(v) = \top$ for every $k \geq 0$.
20. By Def. 6 we have $\mathcal{W} \models (\Box t_v \vee \Box \neg t_v)$ and by Def. 7, $\langle M, 0 \rangle \models t_v$ iff $f(v) = \top$. Thus, $\langle M, k \rangle \models t_v$ iff $f(v) = \top$ for every $k \geq 0$.
21. By 19 and 20 we conclude that Theorem 1 holds for $v \in V(\mathsf{input})$ and $k \geq 0$.

Proof. Proof of Theorem 1 part (B): $v \in V$ and $k = 0$.

22. Consider a logical diagram $LD = \langle V, E, \mathcal{O} \rangle$ and an initializing function f and the *LTL* theories $\mathcal{T}_{LD}$ and $\mathcal{T}_f$. Let $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_f$ and $v \in V$.
23. We assume that $v \in V(\mathsf{M})$.
24. Based on Def. 3, we have $LD_f(v, 0) = \top$ iff $f(v) = \top$.
25. By Def. 7, $\langle \mathcal{W}, 0 \rangle \models t_v \leftrightarrow f(v)$. Consequently, $\langle \mathcal{W}, 0 \rangle \models t_v$ iff $f(v) = \top$.
26. Based on 24 and 25 we conclude that $LD_f(v, 0) = \top$ iff $\langle \mathcal{W}, 0 \rangle \models t_v$. Thus, the theorem holds for $k = 0$ and for every $v \in V(\mathsf{M})$.
27. Let s be a sequence of vertices $s = v_0, \ldots, v_n \in V$ with $v_i \rightarrow v_{i+1} \in E$ for every $i \in 0, \ldots, n-1$ and $u \rightarrow v_0 \notin E$ for every $u \in V$. From Def. 1, we prove that $v_0 \in V(\mathsf{input}) \cup V(\mathsf{M})$. We define the position of a vertex v in a logical diagram as follows: if $v \in V(\mathsf{input}) \cup V(\mathsf{M})$ then $\textsc{pos}(v) = 0$. Otherwise, $\textsc{pos}(v) = max(\{pos(u) \mid u \rightarrow v \in E\}) + 1$.
28. We prove Theorem 1 for $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$ and $k = 0$.

29. We use the proof by induction. Base case: $\text{POS}(v) = 1$.
30. We assume that $v \in V(\mathsf{output})$ and $u \to v \in E$:
    30.a By Def. 3, we have $LD_f(v,0) = ev(LD_f(u,0))$.
    30.b By Def. 6, we have $\mathcal{W} \models \Box(t_u \leftrightarrow t_v)$. Thus, $\langle \mathcal{W}, 0\rangle \models t_v$ iff $\langle \mathcal{W}, 0\rangle \models t_u$.
    30.c As per 27, we have $\text{POS}(u) = 0$ and $u \in V(\mathsf{input}) \cup V(\mathsf{M})$.
    30.d As stated in (A) and 26, the theorem is true for $k = 0$ and for every $w \in V(\mathsf{input}) \cup V(\mathsf{M})$.
    30.e By 30.c and 30.d we have $\langle \mathcal{W}, 0\rangle \models t_u$ iff $LD_f(u,0) = \top$.
    30.f From 30.a, 30.b and 30.e we conclude that $\langle \mathcal{W}, 0\rangle \models t_v$ iff $LD_f(v,0) = \top$. Consequently, Theorem 1 holds for $\text{POS}(v) = 1$, $k = 0$ and $v \in V(\mathsf{output})$.
31. Similarly, we prove that Theorem 1 holds for $k = 0$ and $\text{POS}(v) = 1$ for the vertices $v \in V(\mathsf{not})$, $v \in V(\mathsf{or})$ and $v \in V(\mathsf{and})$.
32. Now, we assume that Theorem 1 holds up to some position $\text{POS}(v) = j$ with $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$. We prove that it also holds for the position $j + 1$.
33. We assume that $v \in V(\mathsf{output})$ with $\text{POS}(v) = j + 1$ and $u \to v \in E$:
    33.a By Def. 3, we have $LD_f(v,0) = ev(LD_f(u,0))$.
    33.b By Def. 6, we have $\mathcal{W} \models \Box(t_u \leftrightarrow t_v)$. Thus, $\langle \mathcal{W}, 0\rangle \models t_v$ iff $\langle \mathcal{W}, 0\rangle \models t_u$.
    33.c As per 27, we have $\text{POS}(u) = j$. Thus, based on 32 we have $\langle M, 0\rangle \models t_u$ iff $LD_f(u,0) = \top$.
    33.d By 33.a, 33.b and 33.c, we conclude that Theorem 1 is true for the position $\text{POS}(v) = j + 1$ with $k = 0$ and $v \in V(\mathsf{output})$.
34. Similarly, we prove that the theorem holds for $k = 0$ and $\text{POS}(v) = j + 1$ for the vertices $v \in V(\mathsf{not})$, $v \in V(\mathsf{or})$ and $v \in V(\mathsf{and})$.
35. We conclude that Theorem 1 is true for $k = 0$ and $v \in v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$.
36. **Conclusion**: by (A), 26 and 35; Theorem 1 holds for $k = 0$ and $v \in V$.

Proof. Proof of Theorem 1 part (C): $v \in V$ and $k \geq 1$.

37. Consider a logical diagram $LD = \langle V, E, \mathcal{O}\rangle$ and an initializing function f. The *LTL* theories are $\mathcal{T}_{LD}$ and $\mathcal{T}_f$. Let $\mathcal{W} \models \mathcal{T}_{LD} \cup \mathcal{T}_f$ and $v \in V$
38. We prove Theorem 1 for $k \geq 1$ by induction. We first prove it holds for the base case $k = 1$ for $v \in V(\mathsf{M}_E)$ then for $v \in V(\mathsf{M}_H)$ and finally for $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$.
39. Let $v \in V(\mathsf{M}_E)$ with $e \xrightarrow{\text{s}} v \in E$, $h \xrightarrow{\text{r}} v \in E$, $e \in V$ and $h \in V$.
40. Let $k = 1$. If $\mathcal{O}(v) = 1$ then $k = 1 \in \mathsf{turn}_{\mathcal{O}}(v)$:
    40.a By Def. 6, $\mathcal{W} \models \Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge ((t_v \wedge \neg t_h) \vee t_e))\big)\Big)$.
    40.b By Lemma 1 we have $\langle \mathcal{W}, 0\rangle \models t_1$.
    40.c By 40.a and 40.b we have $\langle \mathcal{W}, 1\rangle \models t_v$ iff $\langle \mathcal{W}, 0\rangle \models \big((t_v \wedge \neg t_h) \vee t_e\big)$.
    40.d By Def. 3, we have $LD_f(v,1) = \mathsf{ev}((LD_f(v,0) \wedge \neg LD_f(h,0)) \vee LD_f(e,0))$.
    40.e By 40.c and 40.d and knowing that Theorem 1 is true for $k = 0$ (proven in (B)), we conclude that $\langle \mathcal{W}, 1\rangle \models t_v$ iff $LD_f(v,1) = \top$.
41. Let $k = 1$. If $\mathcal{O}(v) \neq 1$ (i.e $k \notin \mathsf{turn}_{\mathcal{O}}(v)$):
    41.a By Def. 6, $\mathcal{W} \models \Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge ((t_v \wedge \neg t_h) \vee t_e))\big)\Big)$.
    41.b By Lemma 1 we have $\langle \mathcal{W}, 0\rangle \not\models t_1$.

41.c By 41.a and 41.b we have $\langle \mathcal{W}, 1\rangle \models t_v$ iff $\langle \mathcal{W}, 0\rangle \models t_v$.
41.d By Def. 3, we have $LD_f(v,1) = \mathsf{ev}(LD_f(v,0))$.
41.e By 41.c and 41.d and knowing that Theorem 1 is true for $k=0$ (proven in (B)), we conclude that $\langle \mathcal{W}, 1\rangle \models t_v$ iff $LD_f(v,1) = \top$.
42. By 40.e and 41.e we conclude that Theorem 1 holds for $k=1$ and $v \in V(\mathsf{M}_E)$.
43. In the same way we prove that Theorem 1 holds for $k=1$ and $v \in V(\mathsf{M}_H)$.
44. Similarly to (B) from 27 to 35, we have Theorem 1 holds for $k=1$ and $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$.
45. By (A), 42, 43 and 44, Theorem 1 holds for $k=1$ and $v \in V$.
46. We assume that Theorem 1 holds for some $k \geq 1$. We prove that it also holds for $k+1$ for $v \in V(\mathsf{M}_E)$ then for $v \in V(\mathsf{M}_H)$ and finally for $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$.
47. Let $v \in V(\mathsf{M}_E)$ with $e \xrightarrow{\mathrm{s}} v \in E$, $h \xrightarrow{\mathrm{r}} v \in E$, $e \in V$ and $h \in V$.
48. Let $k+1 \in \mathsf{turn}_{\mathcal{O}}(v)$:
48.a By Def. 6, $\mathcal{W} \models \Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge ((t_v \wedge \neg t_h) \vee t_e))\big)\Big)$.
48.b By Lemma 1 we have $\langle \mathcal{W}, k\rangle \models t_{\mathcal{O}(v)}$.
48.c By 48.a and 48.b we have $\langle \mathcal{W}, k+1\rangle \models t_v$ iff $\langle \mathcal{W}, k\rangle \models \big((t_v \wedge \neg t_h) \vee t_e\big)$.
48.d Based on Def. 3, $LD_f(v,k+1) = \mathsf{ev}((LD_f(v,k) \wedge \neg LD_f(h,k)) \vee LD_f(e,k))$.
48.e By 48.c and 48.d and knowing that Theorem 1 holds for k as stated in 46, we conclude that $\langle \mathcal{W}, k+1\rangle \models t_v$ iff $LD_f(v,k+1) = \top$.
49. Let $k+1 \notin \mathsf{turn}_{\mathcal{O}(v)}$:
49.a By Def. 6, $\mathcal{W} \models \Box\Big(\bigcirc t_v \leftrightarrow \big((\neg t_{\mathcal{O}(v)} \wedge t_v) \vee (t_{\mathcal{O}(v)} \wedge ((t_v \wedge \neg t_h) \vee t_e))\big)\Big)$.
49.b By Lemma 1 we have $\langle \mathcal{W}, k\rangle \not\models t_{\mathcal{O}(v)}$.
49.c By 49.a and 49.b we have $\langle \mathcal{W}, k+1\rangle \models t_v$ iff $\langle \mathcal{W}, k\rangle \models t_v$.
49.d Based on Def. 3, we have $LD_f(v,k+1) = \mathsf{ev}(LD_f(v,k))$.
49.e By 49.c and 49.d and knowing that Theorem 1 holds for k as stated in 46, we conclude that $\langle \mathcal{W}, k+1\rangle \models t_v$ iff $LD_f(v,k+1) = \top$.
50. By 48.e and 49.e we conclude that Theorem 1 holds for $k+1$ and $v \in V(\mathsf{M}_E)$.
51. Likewise, Theorem 1 holds for $k+1$ and $v \in V(\mathsf{M}_H)$.
52. Similarly to (B) from 27 to 35, Theorem 1 holds for $k+1$ and $v \in V(\mathsf{not}) \cup V(\mathsf{or}) \cup V(\mathsf{and}) \cup V(\mathsf{output})$.
53. By (A), 50, 51 and 52, Theorem 1 holds for every $k \geq 1$ and $v \in V$.

Conclusion: theorem 1 holds for every $k \geq 0$ and for every $v \in V$.

5 Evaluation and discussion

In this section, we evaluate different *LTL* model checking *(MC)* techniques on a real world logical diagram used for a logical controller in a nuclear power plant. The goal is to generate scenarios that could be used for validation tests. These scenarios consist in a set of input values and the set of expected output values. In the testing process, the generated inputs are applied on the logical controller in order to observe the real output values and make sure that they are conform to the expected output values of the generated scenario. However, before

Table 1. Evaluation of the *MC* techniques on $\mathcal{P}_{act}$ and $\mathcal{P}_{deact}$ of all the 12 outputs. m: minutes; s: seconds; mean: median time of the 12 outputs; min/max : minimum/maximum time out of the 12 outputs; (n): n is the length of the counterexample.

	BDD	BMC	SBMC	k-MC
$\mathcal{P}_{act}(o)$	timeout	mean: 26s (33) min: 4.2s (22) max: 2m 11s (57)	mean: 3.4s (33) min: 1.3s (22) max: 11.3s (57)	mean: 6.2s (33) min: 3.9s (24) max: 12.9s (57)
$\mathcal{P}_{deact}(o)$	timeout	mean: 18.2s (32) min: 4s (22) max: 35s (39)	mean: 2.9s (32) min: 1.3s (22) max: 4.4s (39)	mean: 5.3s (32) min: 3.4s (22) max: 7.5s (35)

generating these scenarios it is important to ensure that the logical diagram satisfies the stability property. An unstable behavior is when a value of one or many outputs keep changing indefinitely for fixed inputs. In this case, the logical diagram has to be revised to satisfy the stability. This is because the unstable behavior means that the expected value of the output is not defined and therefore, generating tests for that output becomes meaningless. Verification of the stability and generating test scenarios are the main focus of this evaluation.

The real world logical diagram chosen for the evaluation is representative of many other logical diagrams used by EDF. It contains 16 input vertices, 12 output vertices, 19 memory vertices and 77 vertices of the types $\{\mathsf{and}, \mathsf{or}, \mathsf{not}\}$. First, the satisfaction of the stability property is verified. Then, the purpose is to generate scenarios that set an output to the value true when it is initially set to false (i.e. activation scenarios) and others that put it to false when it is initially set to true (i.e. deactivation scenarios). These scenarios will be generated for each different output. The concerned real world logical diagram will be referred to as LD_R. The results previously established in this paper were implemented to generate the *LTL* theory $\mathcal{T}_{LD_R}$ that encodes the logical diagram LD_R as well as the theory $\mathcal{T}_{stable}$ encoding the stability of LD_R and the theories $\mathcal{T}^{o_i}_{act}$ and $\mathcal{T}^{o_i}_{deact}$ for the activation and deactivation of every output vertex o_i.

The *LTL* theory of stability is $\mathcal{T}_{stable} = \{\Diamond(\Box t_{m_i} \vee \Box\neg t_{m_i}) \mid i \in 1, 2, \ldots, 19\}$. We use the *LTL* model checking techniques to prove the truthfulness of the following *LTL* property: $\mathcal{T}_{LD} \models \mathcal{T}_{stable}$. This property will be referred to as the stability property $\mathcal{P}_{stab}$. If it is proven to be true, then LD_R is uniformly stable. If it is not, then we get a counterexample, i.e. a word $\mathcal{W}$ that is a model of $\mathcal{T}_{LD_R}$ and the theory $\mathcal{T}_f$ of some initializing function f for which LD_R is unstable. The *LTL* theory that encodes the activation of an output o_i which is initially deactivated is $\mathcal{T}^{o_i}_{act} = \{(t_{o_i} \leftrightarrow \bot), (\Diamond\Box t_{o_i})\}$. Likewise, the *LTL* theory that encodes the deactivation of an output o_i which is initially activated is $\mathcal{T}^{o_i}_{deact} = \{(t_{o_i} \leftrightarrow \top), (\Diamond\Box\neg t_{o_i})\}$. In order to generate activation and deactivation scenarios for an output o_i, we apply the model checking on the property $\mathcal{T}_{LD} \cup \mathcal{T}^{o_i}_{act} \models \bot$ referred to as $\mathcal{P}_{act}(o_i)$ and the property $\mathcal{T}_{LD} \cup \mathcal{T}^{o_i}_{deact} \models \bot$ referred to as $\mathcal{P}_{deact}(o_i)$. If $\mathcal{P}_{act}(o_i)$ (resp. $\mathcal{P}_{deact}(o_i)$) is true, then no activation (resp. deactivation) scenarios exist. Otherwise, if $\mathcal{P}_{act}(o_i)$ (resp. $\mathcal{P}_{deact}(o_i)$) is not satisfied we get a counterexample, i.e a word $\mathcal{W}$ that satisfies all the formulae of

the LD_R theory $\mathcal{T}_{LD_R}$, the activation theory $\mathcal{T}_{act}^{o_i}$ (resp. the deactivation theory $\mathcal{T}_{deact}^{o_i}$) and the theory $\mathcal{T}_f$ of some initializing function f for which the output o_i is activated (resp. deactivated). The generated word expresses the testing scenario to be applied on the logical controller. We check $\mathcal{P}_{act}(o_i)$ and $\mathcal{P}_{deact}(o_i)$ properties for every output vertex o_i of the 12 outputs in LD_R.

For this evaluation, the verification of each property was done using a well known symbolic model checking tool called *NuSMV* [5]. Different techniques were used for each property. In the following, we list the evaluated *NuSMV LTL* model checking techniques presented in [4]:

- The Binary Decision Diagram (BDD) based *LTL* Model Checking [7].
- Bounded Model Checking (*BMC*) based on SAT solvers as described in [1].
- Simple Bounded Model Checking (SBMC) based on SAT solvers as in [9].

The usage of the *BDD* based technique allows to verify whether an *LTL* property is true or false and to generate a counterexample when it is false. On the other hand, *BMC* based techniques prove that a property is false by increasingly exploring the different lengths of counterexamples starting from zero to a preset upper bound. When the maximum bound is reached and no counterexamples are found, then the truth of the property is not decided. In other words, unlike the *BDD* based technique, the *BMC* based ones are useful only in case the property is false. In this evaluation, the upper bound of the *BMC* based techniques was set to 1000. Another technique tested for the verification of the properties is the *k-liveness* algorithm based model checking technique (*k-MC*) as described in [6]. The technique was introduced in another tool called *nuXmv* [3]: an extension for *NuSMV*. The usage of the technique is presented in [2]. For this evaluation, the timeout delay for each of the mentioned techniques was set to two hours.

The verification of the uniform stability of LD_R was done by checking the property $\mathcal{P}_{stab}$ using the different techniques. The *BDD*, *BMC*, and *SBMC* techniques timed-out. The stability could not therefore be concluded with these techniques. However, the *k-MC* technique terminated successfully in 26 minutes declaring that the property $\mathcal{P}_{stab}$ is true. Therefore, the logical diagram LD_R is uniformly stable. Each technique was then tested for the activation ($\mathcal{P}_{act}(o_i)$) and deactivation ($\mathcal{P}_{deact}(o_i)$) property for every output o_i. Except for the *BDD* technique which timed-out in every single check, all the other techniques successfully generated counterexamples of the same length for every output of LD_R. The evaluation results are given by table 1. The *SBMC* technique was the fastest to generate the counterexample for every output while *BMC* was considerably slower especially on the longest counterexamples.

This study shows that the proposed *LTL* encoding of the logical diagrams is a good candidate for test generation, as multiple counterexamples can be generated starting from different initial configuration defined by the verification process. We tested the proposed *LTL* encoding on a real, representative logical diagram by comparing different *LTL* model checking techniques. It turns out that *k-MC* and *SBMC* techniques are particularly efficient for checking the properties we defined, allowing us to test the controller's outputs with respect to the LD.

Acknowledgments

David Carral is funded by the ANR project CQFD (ANR-18-CE23-0003).

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems: 5th International Conference, TACAS'99 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99 Amsterdam, The Netherlands, March 22–28, 1999 Proceedings 5. pp. 193–207. Springer (1999)
2. Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: nuXmv 2.0. 0 User Manual. Fondazione Bruno Kessler, Tech. Rept., Trento, Italy (2019)
3. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26. pp. 334–342. Springer (2014)
4. Cavada, R., Cimatti, A., Jochim, C.A., Keighren, G., Olivetti, E., Pistore, M., Roveri, M., Tchaltsev, A.: Nusmv 2.4 user manual. CMU and ITC-irst (2005)
5. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14. pp. 359–364. Springer (2002)
6. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 52–59. IEEE (2012)
7. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods in System Design **10**, 47–71 (1997)
8. Enoiu, E.P., Čaušević, A., Ostrand, T.J., Weyuker, E.J., Sundmark, D., Pettersson, P.: Automated test generation using model checking: an industrial evaluation. International Journal on Software Tools for Technology Transfer **18**, 335–353 (2016)
9. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: Efficient bounded model checking for past LTL. In: Verification, Model Checking, and Abstract Interpretation: 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005. Proceedings 6. pp. 380–395. Springer (2005)
10. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
11. Provost, J., Roussel, J.M., Faure, J.M.: Translating Grafcet specifications into Mealy machines for conformance test purposes. Control Engineering Practice **19**(9), 947–957 (2011)
12. Sfar, A., Irofti, D., Croitoru, M.: A graph based semantics for Logical Functional Diagrams in power plant controllers. In: Foundations of Information and Knowledge Systems: 12th International Symposium, FoIKS 2022, Helsinki, Finland, June 20–23, 2022, Proceedings. pp. 55–74. Springer (2022)
13. Springintveld, J., Vaandrager, F., D'Argenio, P.R.: Testing timed automata. Theoretical computer science **254**(1-2), 225–257 (2001)