Centre Fédéré en Vérification

Technical Report number 2009.110

An Antichain Algorithm for LTL Realizability

Emmanuel Filiot, Naiyong Jin, Jean-François Raskin

This work was partially supported by a FRFC grant: 2.4530.02
and by the MoVES project. MoVES (P6/39) is part of the IAP-Phase VI Interuniversity
Attraction Poles Programme funded by the Belgian State, Belgian Science Policy

http://www.ulb.ac.be/di/ssd/cfv

An Antichain Algorithm for LTL Realizability

Emmanuel Filiot Naiyong Jin Jean-François Raskin

Université Libre de Bruxelles (ULB), Belgium

Abstract. In this paper, we study the structure of underlying automata based constructions for solving the LTL realizability problem. We show how to reduce the LTL realizability problem to a game with an observer which checks that the game visits a bounded number of times accepting states of a universal co-Büchi word automaton. We show that such an observer can be made deterministic and that this deterministic observer has a nice structure which can be exploited by an incremental algorithm that manipulates antichains of game positions. We have implemented this new algorithm and our first results are very encouraging.

1 Introduction

Automata theory has revealed very elegant for solving verification and synthesis problems. A large body of results in computer aided verification can be phrased and solved in this framework. Tools that use those results have been successfully used in industrial context, see [16] for an example. Nevertheless, there are still plenty of research to do and new theory to develop in order to obtain more efficient algorithms able to handle larger or broader classes of practical examples. Recently, we and other authors have shown in [4–6, 14, 20] that several automata-based constructions enjoy structural properties that can be exploited to improve algorithms on automata. For example, in [6] we show how to solve more efficiently the language inclusion problem for nondeterministic Büchi automata by exploiting a partial-order that exists on the state spaces of subset constructions used to solve this problem. Other structural properties have been additionally exploited in [7]. In this paper, we pursue this line of research and revisit the automata-based approach to LTL realizability. We show that there are also automata structures equipped with adequate partial-orders that can be exploited to obtain a more practical decision procedure for this 2EXPTIME-COMPLETE problem.

The realizability problem for an LTL formula ϕ is best seen as a game between two players [13]. Each of the player is controlling a subset of the set P of propositions on which the LTL formula ϕ is constructed. The set of propositions P is partitioned into I the set of *input signals* that are controlled by "Player input" (the environment, also called Player I), and O the set of *output signals* that are controlled by "Player output" (the controller, also called Player O). The realizability game is played in turns. Player O is the protagonist, she wants to satisfy the formula ϕ, while Player I is the antagonist as he wants to falsify the formula ϕ. Player O starts by giving a subset o_0 of propositions [1], Player I responds by giving a subset of propositions i_0, then Player O gives o_1 and Player I responds by i_1, and so on. This game lasts forever and the outcome of the game is the infinite word $w = (i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2) \cdots \in (2^P)^\omega$. We say that

[1] Technically, we could have started with Player I, for modeling reason it is conservative to start with Player O.

Player O wins if the resulting infinite word w is a model of ϕ.[2] This problem is central when dealing with specifications for reactive systems. In that context, the signals of the environment being uncontrollable, unrealizable specifications are useless as they can not be implemented. The LTL realizability problem has been studied starting from the end of the eighties with the seminal works by Pnueli and Rosner [13], and Abadi, Lamport and Wolper [1]. The 2EXPTIME lower bound was established in [15]. [3]

The classical automata-based solution to LTL synthesis can be summarized as follows. Given an LTL formula ϕ, construct a nondeterministic Büchi automaton A_ϕ that accepts all models that satisfy ϕ, transform A_ϕ into a deterministic Rabin automaton B using Safra's determinization procedure [17], and use B as an observer in a turn-based two-player game. Unfortunately, this theoretically elegant procedure has turn out to be very difficult to implement. There are two main reasons for that. First, Safra's determinization procedure generates very complex state spaces: states are colored trees of subsets of states of the original automaton. No nice symbolic data-structure is known to handle such state spaces. Second, the game to solve as the last step (on a potentially doubly-exponential state-space) is a Rabin game, and this problem is known to be NP complete[4].

This situation has triggered further research for alternative procedures. Most notably, Kupferman and Vardi in [10] have recently proposed procedures that avoid the determinization step and so the Safra's construction[5]. In particular, they reduce the LTL realizability problem to the emptiness of a Universal Co-Büchi Tree automaton (UCT). They show how to test emptiness of a UCT by translation to an alternating weak Büchi tree automaton, again translated into a non-deterministic Büchi tree automaton for which testing emptiness is easy. All these steps have been implemented and optimized in several ways by Jobstmann and Bloem in a tool called Lily [9]. In contrast to this method, we will see that our approach is more direct and even without further optimizations, our implementation Acacia outperforms Lily.

In this paper, we also rely on a Safraless decision procedure for the LTL realizability problem and we identify structural properties that allow us to define an antichain algorithm in the line of our previous works. Our procedure uses Universal Co-Büchi Word automaton, UCW. Those automata have the following simple nice property. If a Moore machine M with m states defines a language included into the language defined by a UCW with n states, then obviously every run on the words generated by M contains at most $2mn$ accepting states. As a consequence a Moore machine that enforces a language defined by a UCW also enforces a stronger requirement defined by the same automaton where the acceptance condition is strengthened to a so called $2mn$-bounded one: ”a run is accepting if it passes at most $2mn$ times by an accepting state”. Using the result by Safra, we know that the size of a Moore machine that realizes a language defined by a UCW can be bounded. This gives a reduction from the general problem to

[2] As a subproblem, if the specification is realizable, we want to construct a Moore-machine that implements the winning strategy of the protagonist. This is called the LTL synthesis problem.

[3] Older works also consider the realizability problem but for more expressive and computationally intractable formalisms, see [19].

[4] Instead of Rabin automata, Parity automata can also be used [12]. Nevertheless, there are no known polynomial time algorithm to solve parity games.

[5] As a consequence, they call their new procedures *Safraless* procedures. Nevertheless they use the result by Safra in their proof of correctness.

the problem of the realizability of a k-bounded UCW specification. Contrary to general UCW specifications, k-bounded UCW specifications can easily be made deterministic and, most importantly the underlying deterministic automaton is always equipped with a partial-order on states that can be used to efficiently manipulate its state space using our antichain method. We have implemented this new antichain algorithm and our experiments show promising results.

The rest of this paper is organized as follows. In Section 2, we recall definitions. In Section 3, we show how to reduce the LTL realizability problem to the realizability of a k-bounded UCW specification. In Section 4, we show structural properties of the deterministic structure that we obtain from the k-bounded UCW specification and study antichains for manipulating sets of states of this deterministic structure. In Section 5, we report on preliminary experiments using our antichain algorithm for synthesis and compare them to the results obtained by using the tool Lily [9]. In Section 6, we draw conclusions and identify future works. Detailed proofs are available in Appendix.

2 LTL and Realizability Problem

*Linear Temporal Logic (*LTL*)* The formulas of LTL are defined over a set of atomic propositions P. The syntax is given by the grammar:

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{X}\phi \mid \phi\mathcal{U}\phi \qquad\qquad p \in P$$

The notations true, false, $\phi_1 \wedge \phi_2$, $\Diamond\phi$ and $\Box\phi$ are defined as usual. In particular, $\Diamond\phi = \mathsf{true}\mathcal{U}\phi$ and $\Box\phi = \neg\Diamond\neg\phi$. LTL formulas ϕ are interpreted on infinite words $w = \sigma_0\sigma_1\sigma_2\cdots \in (2^P)^\omega$ via a satisfaction relation $w \models \phi$ inductively defined as follows: (i) $w \models p$ if $p \in \sigma_0$, (ii) $w \models \phi_1 \vee \phi_2$ if $w \models \phi_1$ or $w \models \phi_2$, (iii) $w \models \neg\phi$ if $w \not\models \phi$, (iv) $w \models \mathcal{X}\phi$ if $\sigma_1\sigma_2 \ldots \models \phi$, and (v) $w \models \phi_1\mathcal{U}\phi_2$ if there is $n \geq 0$ such that $\sigma_n\sigma_{n+1}\ldots \models \phi_2$ and for all $0 \leq i < n$, $\sigma_i\sigma_{i+1}\ldots \models \phi_1$.

LTL *Realizability Problem* As recalled in the introduction, the realizability problem for LTL is best seen as a game between two players. Each of the player is controlling a subset of the set P of propositions on which the LTL formula is constructed. Accordingly, unless otherwise stated, we partition the set of propositions P into I the set of *input signals* that are controlled by ”Player input” (the environment, also called Player I), and O is the set of *output signals* that are controlled by ”Player output” (the controller, also called Player O). It is also useful to associate this partition of P with the three following alphabets: $\Sigma = 2^P$, $\Sigma_I = 2^I$, and $\Sigma_O = 2^O$. We denote by $\varnothing$ the empty set. The realizability game is played in turns. Player O starts by giving a subset o_0 of propositions, Player I responds by giving a subset of propositions i_0, then Player O gives o_1 and Player I responds by i_1, and so on. This game lasts forever and the output of the game is the infinite word $(i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2)\cdots \in \Sigma^\omega$. The players play according to strategies. A strategy for Player O is a (total) mapping $\lambda_O : (\Sigma_O\Sigma_I)^* \rightarrow \Sigma_O$ while a strategy for Player I is a (total) mapping $\lambda_I : \Sigma_O(\Sigma_I\Sigma_O)^* \rightarrow \Sigma_I$. The outcome of the strategies λ_O and λ_I is the word $\mathsf{outcome}(\lambda_O, \lambda_I) = (o_0 \cup i_0)(o_1 \cup i_1)\ldots$ such that for all $j \geq 0$, $o_j = \lambda_O(o_0 i_0 \ldots o_{j-1} i_{j-1})$ and $i_j = \lambda_I(o_0 i_0 o_{j-1} i_{j-1} o_j)$. In particular, $o_0 = \lambda_O(\epsilon)$ and $i_0 = \lambda_I(o_0)$.
We can now define the realizability problem. Given an LTL formula ϕ (the specification), the *realizability problem* is to decide whether there exists a strategy λ_O of Player

O such that for all strategies λ_I of Player I, $\mathsf{outcome}(\lambda_O, \lambda_I) \models \phi$. If such a strategy exists, we say that the specification ϕ is *realizable*.
For example, let $I = \{q\}$, $O = \{p\}$ and $\psi = p\,\mathcal{U}q$. The formula ψ is not realizable. If up to the instant for output, the environment never asserts q, then the controller has to assert p as it has no idea of whether the environment will assert p in the future. This strategy will not make the property $p\,\mathcal{U}q$ satisfied provided the environment never asserts q. However $\Diamond q \rightarrow (p\,\mathcal{U}q)$ is realizable. The assumption $\Diamond q$ states that q will hold at some point, and ensures the controller wins if it always asserts p.

Infinite Word Automata An *infinite word automaton* over the finite alphabet Σ is a tuple $A = (\Sigma, Q, q_0, \alpha, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\alpha \subseteq Q$ is a set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation. For all $q \in Q$ and all $\sigma \in \Sigma$, we let $\delta(q, \sigma) = \{q' \mid (q, \sigma, q') \in \delta\}$. We let $|A| = |Q| + |\Delta|$ be the size of A. We say that A is *deterministic* if $\forall q \in Q\ \forall \sigma \in \Sigma\ |\delta(q, \sigma)| \leq 1$. It is *complete* if for all $q \in Q$, and all $\sigma \in \Sigma$, $\delta(q, \sigma) \neq \varnothing$. In this paper, unless otherwise stated, the automata are complete. A *run* of A on a word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho_0\rho_1 \cdots \in Q^\omega$ such that $\rho_0 = q_0$ and for all $i \geq 0$, $q_{i+1} \in \delta(q_i, \rho_i)$. We denote by $\mathsf{Runs}_A(w)$ the set of runs of A on w. We denote by $\mathsf{Visit}(\rho, q)$ the number of times the state q occurs along the run ρ. We consider three acceptance conditions (a.c.) for infinite word automata. A word w is accepted by A if (depending on the a.c.):

$$\begin{array}{ll} \text{Non-deterministic Büchi} & : \exists \rho \in \mathsf{Runs}_A(w),\ \exists q \in \alpha,\ \mathsf{Visit}(\rho, q) = \infty \\ \text{Universal Co-Büchi} & : \forall \rho \in \mathsf{Runs}_A(w),\ \forall q \in \alpha,\ \mathsf{Visit}(\rho, q) < \infty \\ \text{Universal } K\text{-Co-Büchi} & : \forall \rho \in \mathsf{Runs}_A(w),\ \forall q \in \alpha,\ \mathsf{Visit}(\rho, q) \leq K \end{array}$$

The set of words accepted by A with the non-deterministic Büchi a.c. is denoted by $L_{\text{b}}(A)$, and with this a.c. in mind, we say that A is a non-deterministic Büchi word automaton, NBW for short. Similarly, we denote respectively by $L_{\text{uc}}(A)$ and $L_{\text{uc},K}(A)$ the set of words accepted by A with the universal co-Büchi and universal K-co-Büchi a.c. respectively. With those interpretations, we say that A is a universal co-Büchi automaton (UCW) and that (A, K) is a universal K-co-Büchi automaton ($\mathsf{U}K\mathsf{CW}$) respectively. By duality, we have clearly $L_{\text{b}}(A) = \overline{L_{\text{uc}}(A)}$, for any infinite word automaton A. Finally, note that for any $0 \leq K_1 \leq K_2$, we have that $L_{\text{uc},K_1}(A) \subseteq L_{\text{uc},K_2}(A) \subseteq L_{\text{uc}}(A)$.

Infinite automata and LTL It is well-known (see for instance [18]) that NBWs subsume LTL in the sense that for all LTL formula ϕ, there is an NBW A_ϕ (possibly exponentially larger) such that $L_{\text{b}}(A_\phi) = \{w \mid w \models \phi\}$. Similarly, by duality it is straightforward to associate an equivalent UCW to any LTL formula ϕ: take $A_{\neg\phi}$ with the universal co-Büchi a.c., so $L_{\text{uc}}(A_{\neg\phi}) = \overline{L_{\text{b}}(A_{\neg\phi})} = L_{\text{b}}(A_\phi) = \{w \mid w \models \phi\}$.

To reflect the game point of view of the realizability problem, we introduce the notion of turn-based automata to define the specification. A *turn-based automaton* A over the input alphabet Σ_I and the output alphabet Σ_O is a tuple $A = (\Sigma_I, \Sigma_O, Q_I, Q_O, q_0, \alpha, \delta_I, \delta_O)$ where Q_I, Q_O are finite sets of input and output states respectively, $q_0 \in Q_O$ is the initial state, $\alpha \subseteq Q_I \cup Q_O$ is the set of final states, and $\delta_I \subseteq Q_I \times \Sigma_I \times Q_O$, $\delta_O \subseteq Q_O \times \Sigma_O \times Q_I$ are the input and output transition relations respectively. It is *complete* if for all $q_I \in Q_I$, and all $\sigma_I \in \Sigma_I$, $\delta_I(q_I, \sigma_I) \neq \varnothing$, and for all $q_O \in \Sigma_O$ and all $\sigma_O \in \Sigma_O$, $\delta_O(q_O, \sigma_O) \neq \varnothing$. As for usual automata, in this paper we assume that turn-based automata are always complete. Turn-based automata A still runs on words

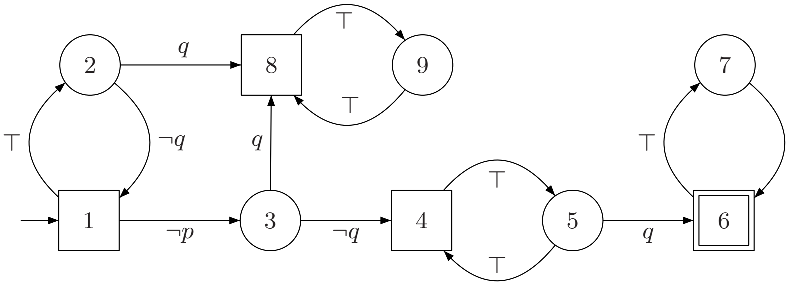

Fig. 1. tbUCW for $\Diamond q \rightarrow (p\mathcal{U}q)$ where $I = \{q\}$ and $O = \{p\}$

from Σ^ω as follows: a run on a word $w = (o_0 \cup i_0)(o_1 \cup i_1) \cdots \in \Sigma^\omega$ is a word $\rho = \rho_0\rho_1 \cdots \in (Q_O Q_I)^\omega$ such that $\rho_0 = q_0$ and for all $j \geq 0$, $(\rho_{2j}, o_j, \rho_{2j+1}) \in \delta_O$ and $(\rho_{2j+1}, i_j, \rho_{2j+2}) \in \delta_I$. All the acceptance conditions considered in this paper carry over to turn-based automata. Turn-based automata with acceptance conditions C are denoted by tbC, e.g. tbNBW. Every UCW (resp. NBW) with state set Q and transition set Δ is equivalent to a tbUCW (resp. tbNBW) with $|Q|+|\Delta|$ states: the new set of states is $Q \cup \Delta$, final states remain the same, and each transition $r = q \xrightarrow{\sigma_o \cup \sigma_i} q' \in \Delta$ where $\sigma_o \in \Sigma_O$ and $\sigma_i \in \Sigma_I$ is split into a transition $q \xrightarrow{\sigma_o} r$ and a transition $r \xrightarrow{\sigma_i} q'$.

Moore Machines It is known that if the LTL specification is realizable, it is also realizable by a finite-state strategy that can be represented for instance by a Moore machine [13]. A *Moore machine* M with input alphabet Σ_I and output alphabet Σ_O is a tuple $(\Sigma_I, \Sigma_O, Q_M, q_0, \delta_M, g_M)$ where Q_M is a finite set of states with initial state q_0, $\delta_M : Q_M \times \Sigma_I \rightarrow Q_M$ is a transition function, and $g_M : Q \rightarrow \Sigma_O$ is an output function. It is *complete* if for all $q \in Q_M$ and all $\sigma \in \Sigma_I$, $\delta_M(q, \sigma)$ is defined. We extend δ_M to $\delta_M^* : \Sigma_I^* \rightarrow Q_M$ inductively as follows: $\delta_M^*(\epsilon) = q_0$ and $\delta_M^*(u\sigma) = \delta_M(\delta_M^*(u), \sigma)$. The language of M, denoted by $L(M)$, is the set of words $w = (o_0 \cup i_0)(o_1 \cup i_1) \cdots \in \Sigma_P^\omega$ such that for all $j \geq 0$, $\delta_M^*(i_0 \ldots i_{j-1})$ is defined and $o_j = g(\delta_M^*(i_0 \ldots i_{j-1}))$. In particular, $o_0 = g(\delta_M^*(\epsilon)) = g(q_0)$. The size of a Moore machine is defined similarly as the size of an automaton.

Thanks to previous remarks, the LTL realizability problem can be reduced to deciding, given a tbUCW A over inputs Σ_I and outputs Σ_O, whether there is a non-empty complete Moore machine M such that $L(M) \subseteq L_{\text{uc}}(A)$.

Running example A tbUCW A equivalent to $\Diamond q \rightarrow (p\mathcal{U}q)$ is depicted in Fig. 1. Output states $Q_O = \{1, 4, 6, 8\}$ are depicted by squares and input states $Q_I = \{2, 3, 5, 7, 9\}$ by circles. In the transitions, $\top$ stands for the sets Σ_I or Σ_O, depending on the context, $\neg q$ (resp. $\neg p$) stands for the sets that do not contain q (resp. p), i.e. the empty set. One can see that starting from state 1, if the controller does not assert p and next the environment does not assert q, then the run is in state 4. From this state, whatever the controller does, if the environment asserts q, then the controller loses, as state 6 will be visited infinitely often. A strategy for the controller is to assert p all the time, therefore the runs will loop in states 1 and 2 until the environment asserts q. Afterwards the runs will loop in states 8 and 9, which are non-final.

3 Reduction to a U*K*CW Objective

In this section, we reduce the realizability problem with a specification given by a turn-based universal co-Büchi automaton (tbUCW) to a specification given by a turn-based universal K-co-Büchi automaton (tbU*K*CW). Then we reduce this new problem to an infinite turn-based two-player game with a safety winning condition. This is done via an easy determinization of tbU*K*CWs (which produces a deterministic tbU*K*CW). To solve this game efficiently, we propose an antichain-based algorithm in Section 4.

Lemma 1. *Let A be a* tbUCW *over inputs Σ_I and outputs Σ_O with n states, and M be a Moore machine over inputs Σ_I and outputs Σ_O with m states. Then $L(M) \subseteq L_{uc}(A)$ iff $L(M) \subseteq L_{uc,2mn}(A)$.*

Proof. The back direction is obvious since $L_{\text{uc},k}(A) \subseteq L_{\text{uc}}(A)$ for all $k \in \mathbb{N}$. We sketch the forth direction (the whole proof is in Appendix). Informally, the infinite paths of M starting from the initial state defines words that are accepted by A. Therefore in the product of M and A, there is no cycle visiting an accepting state of A, which allows one to bound the number of visited final states by the number of states in the product.□

Lemma 2. *Given a realizable* tbUCW *A over inputs Σ_I and outputs Σ_O with n states, if A is realizable, there exists a non-empty and complete Moore machine with at most $n^{2n+2}+1$ states that realizes it.*

Proof. We sketch the proof. In a first step, we show by using Safra's determinization of NBWs that A is equivalent to a turn-based deterministic and complete parity automaton A^d. By using a result of [12], we can assume that A^d has at most $m := 2n^{2n+2}+2$ states. We then view A^d has a turn-based two-player parity game $G(A^d)$ (with at most m states) such that A^d (or equivalently A) is realizable iff Player O has a winning strategy in $G(A^d)$. It is known that parity games admit memoryless strategies [8]. Therefore if A^d is realizable, there exists a strategy for Player O in $G(A^d)$ that can be obtained by removing all but one outgoing edge per Player O's state. We can finally transform this strategy into a Moore machine with at most $n^{2n+2}+1$ states that realizes A^d (and A).

The following Theorem states that we can reduce the realizability of a tbUCW specification to the realizability of a tbU*K*CW specification.

Theorem 1. *Let A be a* tbUCW *over Σ_I, Σ_O with n states and $K = 2n(n^{2n+2}+1)$. Then A is realizable iff (A, K) is realizable.*

Proof. If A is realizable, by Lemma 2, there is a non-empty and complete Moore machine M with m states ($m \leq n^{2n+2}+1$) realizing A. Hence we have $L(M) \subseteq L_{\text{uc}}(A)$ and by Lemma 1, this is equivalent to $L(M) \subseteq L_{\text{uc},2mn}(A)$. We can conclude since $L_{\text{uc},2mn}(A) \subseteq L_{\text{uc},K}(A)$ ($2mn \leq K$). The converse is obvious since $L_{\text{uc},K} \subseteq L_{\text{uc}}(A)$.

In the first part of this section, we reduced the tbUCW realizability problem to the tbU*K*CW realizability problem. In the next part, we reduce this new problem to a safety game. It is based on the determinization of tbU*K*CWs into turn-based deterministic universal 0-Co-Büchi automata, which can also be viewed as safety games.

Safety Game Turn-based two-player games are played on game arenas by two players, Player I and Player O. A *game arena* is a tuple $G = (S_O, S_I, s_0, T)$ where S_I, S_O are disjoint sets of player states (S_I for Player I and S_O for Player O), $s_0 \in S_O$ is the initial state, and $T \subseteq S_O \times S_I \cup S_I \times S_O$ is the transition relation. A *finite play* on G of length n is a finite word $\pi = \pi_0\pi_1 \dots \pi_n \in (S_O \cup S_I)^*$ such that $\pi_0 = s_0$ and for all $i = 0, \dots, n-1$, $(\pi_i, \pi_{i+1}) \in T$. Infinite plays are defined similarly. Note that all infinite plays belong to $(S_0 S_I)^\omega$. A *winning condition* W is a subset of $(S_O S_I)^\omega$. A play π is won by Player O if $\pi \in W$, otherwise it is won by Player I. A *strategy* λ_i for Player i ($i \in \{I, O\}$) is a mapping that maps any finite plays whose last state s is in S_i to a state s' such that $(s, s') \in T$. The *outcome* of a strategy λ_i of Player i is the set $\mathsf{Outcome}_G(\lambda_i)$ of infinite plays $\pi = \pi_0\pi_1\pi_2 \dots \in (S_O S_I)^\omega$ such that for all $j \geq 0$, if $\pi_j \in S_i$, then $\pi_{j+1} = \lambda_i(\pi_0, \dots, \pi_j)$. We consider the safety winning condition. It is given by a subset of states denoted by safe. A strategy λ_i for Player i is *winning* if $\mathsf{Outcome}_G(\lambda_i) \subseteq \mathsf{safe}^\omega$. We sometimes write $(S_O, S_I, s_0, T, \mathsf{safe})$ to denote the game G with safety condition safe. Finally, a strategy λ_i for Player i is winning in the game G *from a state* $s \in S_O \cup S_I$ if it is winning in (S_O, S_I, s, T).

Determinization of UKCW Let A be a tbUCW $(\Sigma_O, \Sigma_I, Q_O, Q_I, q_0, \alpha, \Delta_O, \Delta_I)$ and $K \in \mathbb{N}$. We let $Q = Q_O \cup Q_I$ and $\Delta = \Delta_O \cup \Delta_I$. It is easy to construct an equivalent turn-based deterministic universal 0-co-Büchi automaton $\det(A, K)$. Intuitively, it suffices to extend the usual subset construction with counters, for all $q \in Q$, that count (up to $K + 1$) the maximal number of accepting states which have been visited by runs ending up in q. If the counter of a state is -1, it means that no run on the prefix read so far ends up in q. The final states are the sets in which a state has its counter greater than K. Let $[n] = \{-1, 0, 1, \dots, n\}$, for all $n \in \mathbb{N}$. Formally, we let $\det(A, K) = (\Sigma_O, \Sigma_I, \mathcal{F}_O, \mathcal{F}_I, F_0, \alpha', \delta_O, \delta_I)$ where:

$$\begin{array}{ll}
\mathcal{F}_O & = \{F \mid F \text{ is a mapping from } Q_O \text{ to } [K+1]\} \\
\mathcal{F}_I & = \{F \mid F \text{ is a mapping from } Q_I \text{ to } [K+1]\} \\
F_0 & = q \in Q_O \mapsto \begin{cases} -1 & \text{if } q \neq q_0 \\ (q_0 \in F) & \text{otherwise} \end{cases} \\
\alpha' & = \{F \in \mathcal{F}_I \cup \mathcal{F}_O \mid \exists q, F(q) > K\} \\
\mathsf{succ}(F, \sigma) & = q \mapsto \max\{\min(K+1, F(p) + (q \in \alpha)) \mid q \in \Delta(p, \sigma), F(p) \neq -1\} \\
\delta_O & = \mathsf{succ}|_{\mathcal{F}_O \times \Sigma_O} \qquad \delta_I \;=\; \mathsf{succ}|_{\mathcal{F}_I \times \Sigma_I}
\end{array}$$

where $\max\ \emptyset = -1$, and $(q \in \alpha) = 1$ if q is in α, and 0 otherwise. The automaton $\det(A, K)$ has the following properties:

Proposition 1. *Let A be a tbUCW and $K \in \mathbb{N}$. Then $\det(A, K)$ is deterministic, complete, and $L_{uc,0}(\det(A, K)) = L_{uc,K}(A)$.*

Reduction to a Safety game Finally, we define the game $G(A, K)$ as follows: it is $det(A, K)$ where input states are viewed as Player I's states and output states as Player O's states. Transition labels are ignored. Formally, $G(A, K) = (\mathcal{F}_O, \mathcal{F}_I, F_0, T, \mathsf{safe})$ where $\mathsf{safe} = \mathcal{F} - \alpha'$ and $T = \{(F, F') \mid \exists \sigma \in \Sigma_O \cup \Sigma_I, F' = \mathsf{succ}(F, \sigma)\}$.

Theorem 2 (Reduction to a safety game). *Let A be a tbUCW over inputs Σ_I and outputs Σ_O with n states ($n > 0$), and let $K = 2n(n^{2n+2} + 1)$. The specification A is realizable iff Player O has a winning strategy in the game $G(A, K)$.*

4 Antichain-based Symbolic Algorithm

A fixed point algorithm In the previous section, we have shown how to reduce the realizability problem to a safety game. Symbolic algorithms for solving safety games are constructed using the so-called controllable predecessor operator, see [8] for details. Let $A = (\Sigma_O, \Sigma_I, Q_0, Q_I, q_0, \alpha, \Delta_O, \Delta_I)$ be a tbUCW with n states, $K = 2n(n^{2n+2}+1)$ and $G(A, K) = (\mathcal{F}_O, \mathcal{F}_I, F_0, T, \mathsf{safe})$ be the two-player turn-based safety game defined in the previous section. Remind that $\Delta = \Delta_O \cup \Delta_I$ and let $\mathcal{F} = \mathcal{F}_O \cup \mathcal{F}_I$. In our case, the controllable predecessor operator is based on the two following monotonic functions over $2^{\mathcal{F}}$:

$$\begin{array}{ll} \mathsf{Pre}_I : 2^{\mathcal{F}_O} & \rightarrow 2^{\mathcal{F}_I} \\ \quad S & \mapsto \{F \in \mathcal{F}_I \mid \forall F' \in \mathcal{F}_O, (F, F') \in T \implies F' \in S\} \cap \mathsf{safe} \end{array}$$

$$\begin{array}{ll} \mathsf{Pre}_O : 2^{\mathcal{F}_I} & \rightarrow 2^{\mathcal{F}_O} \\ \quad S & \mapsto \{F \in \mathcal{F}_O \mid \exists F' \in \mathcal{F}_I, (F, F') \in T\} \cap \mathsf{safe} \end{array}$$

Let $\mathsf{CPre} = \mathsf{Pre}_O \circ \mathsf{Pre}_I$ (CPre stands for "controllable predecessors"). The function CPre is monotonic over the complete lattice $(2^{\mathcal{F}_O}, \subseteq)$, and so it has a greatest fixed point that we denote by CPre^*.

Theorem 3. *The set of states from which Player O has a winning strategy in $G(A, K)$ is equal to* CPre^*.

In particular, by Theorem 2, $F_0 \in \mathsf{CPre}^*$ iff the specification A is realizable. To compute CPre^*, we consider the following $\subseteq$-descending chain: $S_0 = \mathcal{F}$, and for $i \geq 0$ $S_{i+1} = \mathsf{CPre}(S_i) \cap S_i$, until $S_{k+1} = S_k$.

Ordering of game configurations. We define the relation $\preceq \subseteq \mathcal{F}_I \times \mathcal{F}_I \cup \mathcal{F}_O \times \mathcal{F}_O$ by $F \preceq F'$ iff $\forall q$, $F(q) \leq F'(q)$. It is clear that $\preceq$ is a partial order. Intuitively, if Player O can win from F' then she can also win from all $F \preceq F'$. Formally, $\preceq$ is a game simulation relation in the terminology of [3].

Closed sets and antichains. A set $S \subseteq \mathcal{F}$ is *closed for* $\preceq$, if $\forall F \in S \cdot \forall F' \preceq F \cdot F' \in S$. We usually omit references to $\preceq$ if clear from the context. Let S_1 and S_2 be two closed sets, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are closed. Furthermore, the image of a closed set S by the functions Pre_I, Pre_O, and CPre are closed sets:

Lemma 3. *For all closed sets $S_1, S_2 \subseteq \mathcal{F}_I$, $S_3 \subseteq \mathcal{F}_O$, the sets* $\mathsf{Pre}_O(S_1)$, $\mathsf{CPre}(S_2)$, *and* $\mathsf{Pre}_I(S_3)$ *are closed.*

As a consequence, all the sets manipulated by the symbolic algorithms are closed sets. We next show how to represent and manipulates those sets efficiently.

The *closure* of a set $S \subseteq \mathcal{F}$, denoted by $\downarrow S$, is the set $S' = \{F' \in \mathcal{F} \mid \exists F \in S \cdot F' \preceq F\}$. Note that for all closed sets $S \subseteq \mathcal{F}$, $\downarrow S = S$. A set $L \subseteq \mathcal{F}$ is an *antichain* if all elements of L are incomparable for $\preceq$. Let $S \subseteq \mathcal{F}$, we denote by $\lceil S \rceil$ the set of maximal elements of S, that is $\lceil S \rceil = \{F \in S \mid \nexists F' \in S \cdot F' \neq F \wedge F \preceq F'\}$, it is an antichain. If S is closed then $\downarrow \lceil S \rceil = S$, i.e. antichains are *canonical representations* for closed sets. Next, we show that antichains are a compact and efficient representation to manipulate closed sets in $\mathcal{F}$. We start with the algorithm for union, intersection, membership and inclusion. Since the size of a state $F \in \mathcal{F}$ is in practice much smaller than the number of states in the antichains, we consider that operations on states are in constant time.

Proposition 2. *Let $L_1, L_2 \subseteq \mathcal{F}$ be two antichains and $F \in \mathcal{F}$, then (i) $\downarrow L_1 \cup \downarrow L_2 = \downarrow\lceil L_1 \cup L_2 \rceil$, this antichain can be computed in time $O((|L_1| + |L_2|)^2)$ and its size is bounded by $|L_1| + |L_2|$, (ii) $\downarrow L_1 \cap \downarrow L_2 = \downarrow\lceil L_1 \sqcap L_2 \rceil$, where $F_1 \sqcap F_2 : q \mapsto \min(F_1(q), F_2(q))$, this antichain can be computed in time $O(|L_1|^2 \times |L_2|^2)$ and its size is bounded by $|L_1| \times |L_2|$, (iii) $\downarrow L_1 \subseteq \downarrow L_2$ iff $\forall F_1 \in L_1 \cdot \exists F_2 \in L_2 \cdot F_1 \preceq F_2$, which can be established in time $O(|L_1| \times |L_2|)$, (iv) $F \in \downarrow L_1$ can be established in time $O(|L_1| \times |Q|)$.*

Let us now turn to the computation of controllable predecessors. Let $F \in \mathcal{F}$, and $\sigma \in \Sigma_I \cup \Sigma_O$. We denote by $\Omega(F, \sigma) \in \mathcal{F}$ the function defined by:

$$\Omega(F, \sigma) : q \in Q \mapsto \min\{\max(-1, F(q') - (q' \in \alpha)) \mid (q, \sigma, q') \in \delta\}$$

Note that since A is complete, the argument of min is a non-empty set. The function Ω is not the inverse of the function succ, as succ has no inverse in general. Indeed, it might be the case that a state $F \in \mathcal{F}$ has no predecessors or has more than one predecessor H such that $\mathsf{succ}(H, \sigma) = F$. However, we prove the following:

Proposition 3. *For all $F, F' \in \mathcal{F} \cap \mathsf{safe}$, and all $\sigma \in \Sigma_I \cup \Sigma_O$, the following propositions hold:*

$$\begin{array}{ll} (i)\ \ F \preceq F' \implies \Omega(F, \sigma) \preceq \Omega(F', \sigma) & (iii)\ F \preceq \Omega(\mathsf{succ}(F, \sigma), \sigma) \\ (ii)\ F \preceq F' \implies \mathsf{succ}(F, \sigma) \preceq \mathsf{succ}(F', \sigma) & (iv)\ \mathsf{succ}(\Omega(F, \sigma), \sigma) \preceq F \end{array}$$

For all $S \subseteq \mathcal{F}$ and $\sigma \in \Sigma_I \cup \Sigma_O$, we denote by $\mathsf{Pre}(S, \sigma) = \{F \mid \mathsf{succ}(F, \sigma) \in S\}$ the set of predecessors of S. The set of predecessors of a closed set $\downarrow F$ is closed and has a unique maximal element $\Omega(F, \sigma)$:

Lemma 4. *For all $F \in \mathcal{F} \cap \mathsf{safe}$ and $\sigma \in \Sigma_I \cup \Sigma_O$, $\mathsf{Pre}(\downarrow F, \sigma) = \downarrow\Omega(F, \sigma)$.*

Proof. Let $H \in \mathsf{Pre}(\downarrow F, \sigma)$. Hence $\mathsf{succ}(H, \sigma) \preceq F$. By Prop. 3$(i)$, we have $\Omega(\mathsf{succ}(H, \sigma), \sigma) \preceq \Omega(F, \sigma)$, from which we get $H \preceq \Omega(F, \sigma)$, by Prop. 3$(iii)$. Conversely, let $H \preceq \Omega(F, \sigma)$. By Prop. 3$(ii)$, $\mathsf{succ}(H, \sigma) \preceq \mathsf{succ}(\Omega(F, \sigma), \sigma)$. Since by Prop. 3$(iv)$, $\mathsf{succ}(\Omega(F, \sigma), \sigma) \preceq F$, we get $\mathsf{succ}(H, \sigma) \preceq F$. □

We can now use the previous result to compute the controllable predecessors:

Proposition 4. *Let A be a tbUKCW. Given two antichains L_1, L_2 such that $L_1 \subseteq \mathcal{F}_I \cap \mathsf{safe}$ and $L_2 \subseteq \mathcal{F}_O \cap \mathsf{safe}$:*

$$\begin{array}{l} \mathsf{Pre}_O(\downarrow L_1) = \bigcup_{\sigma \in \Sigma_O} \mathsf{Pre}(\downarrow L_1, \sigma) = \bigcup_{\sigma \in \Sigma_O} \downarrow\{\Omega(F, \sigma) \mid F \in L_1\} \\ \mathsf{Pre}_I(\downarrow L_2) \ = \bigcap_{\sigma \in \Sigma_I} \mathsf{Pre}(\downarrow L_2, \sigma) = \bigcap_{\sigma \in \Sigma_I} \downarrow\{\Omega(F, \sigma) \mid F \in L_2\} \end{array}$$

$\mathsf{Pre}_O(\downarrow L_1)$ can be computed in time $O(|\Sigma_O| \times |A| \times |L_1|)$, and $\mathsf{Pre}_I(\downarrow L_2)$ can be computed in time $O((|A| \times |L_2|)^{|\Sigma_I|})$.

As stated in the previous proposition, the complexity of our algorithm for computing the Pre_I is worst-case exponential. We establish as a corollary of the next proposition that this is unavoidable unless $P{=}NP$. Given a graph $G = (V, E)$, a set of vertices W is an independent set iff no pairs of elements in W are linked by an edge in E. We denote by $\mathsf{IND}(G) = \{W \subseteq V \mid \forall\{v, v'\} \in E \cdot v \notin W \vee v' \notin W\}$ the set of independent sets of vertices of G. The problem ”independent set” asks given a graph $G = (V, E)$ and an integer $0 \leq k \leq |V|$, if there exists an independent set in G of size larger than k. It is known to be NP-complete.

Proposition 5. *Given a graph $G = (V, E)$, we can construct in deterministic polynomial time a* UKCW *A, with $K = 0$, and an antichain L such that* $\mathsf{IND}(G) = \downarrow \mathsf{Pre}_I(\mathsf{Pre}_O(\mathsf{Pre}_O((L)))$.

Corollary 1. *There is no polynomial time algorithm to compute the* Pre_I *operation on antichains unless $P = NP$.*

Note that this negative result is not a weakness of antichains. Indeed, it is easy to see from the proofs of those results that any algorithm based on a data structure that is able to represent compactly the set of subsets of a given set has this property.

Incremental Algorithm. In practice, for checking the existence of a winning strategy for Player O in the safety game, we rely on an incremental approach. We use the following property of UKCWs: for all $K_1, K_2 \cdot 0 \leq K_1 \leq K_2 \cdot L_{uc,L_1}(A) \subseteq L_{uc,K_2}(A) \subseteq L_{uc}(A)$. So, the following theorem which is a direct consequence of the previous property allows us to test the existence of strategies for increasing values of K:

Theorem 4. *For all* tbUCW*s A, for all $K \geq 0$, if Player O has a winning strategy in the game $G(A, K)$ then the specification defined by A is realizable.*

Unrealizable Specifications. The incremental algorithm is not reasonable to test unrealizability. Indeed, with this algorithm it is necessary to reach the bound $2n(n^{2n+2}+1)$ to conclude for unrealizability. To obtain a more practical algorithm, we rely on the determinacy of ω-regular games (a corollary of the general result by Martin[11]).

Theorem 5. *For all* LTL *formulas ϕ, either (i) there exists a Player O's strategy λ_O s.t. for all Player I's strategies λ_I,* $\mathsf{outcome}(\lambda_O, \lambda_I) \models \phi$*, or there exists a Player I's strategy λ_I s.t. for all Player O's strategies λ_O,* $\mathsf{outcome}(\lambda_O, \lambda_I) \models \neg\phi$.

So, when an LTL specification ϕ is not realizable for Player O, it means that $\neg\phi$ is realizable for Player I. To avoid in practice the enumeration of values for K up to $2n(n^{2n+2}+1)$, we propose the following algorithm. First, given the LTL formula ϕ, we construct two UCWs: one that accepts all the models of ϕ, denoted by A_ϕ, and one that accepts all the models of $\neg\phi$, denoted by $A_{\neg\phi}$. Then we check realizability by Player O of ϕ, and in parallel realizability by Player I of $\neg\phi$, incrementing the value of K. When one of the two processes stops, we know if ϕ is realizable or not. In practice, we will see that either ϕ is not realizable for Player O for a small value of K or $\neg\phi$ is realizable for Player I for a small value of K.

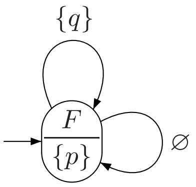

Fig. 2. Moore machine

Example We apply the antichain algorithm on the $\mathtt{tbUCW}$ depicted in Fig. 1, with $K = 1$. Remind that $I = \{q\}$ and $O = \{p\}$, so that $\Sigma_I = \{\varnothing, \{q\}\}$ and $\Sigma_O = \{\varnothing, \{p\}\}$. For space reasons, we cannot give the whole fixpoint computation. We starts with the safe state in $G(A, K)$ for Player O, i.e. the constant function from Q_O to 1 denoted by $F_1 = (1 \mapsto 1, 4 \mapsto 1, 6 \mapsto 1, 8 \mapsto 1)$. It represents the set $\downarrow F_1$. Then we compute $\lceil \mathsf{Pre}_I(\downarrow F_1) \rceil \cap \mathsf{safe} = \lceil \downarrow \Omega(F_1, \{q\}) \cap \ \downarrow \Omega(F_1, \varnothing) \rceil \cap \mathsf{safe}$. We have $\Omega(F_1, \{q\}) = (2 \mapsto 1, 3 \mapsto 1, 5 \mapsto 0, 7 \mapsto 0, 9 \mapsto 1)$ and $\Omega(F_1, \varnothing) = (2 \mapsto 1, 3 \mapsto 1, 5 \mapsto 1, 7 \mapsto 0, 9 \mapsto 1)$. Therefore $\lceil \mathsf{Pre}_I(\downarrow F_1) \rceil = \{F_2 := (2 \mapsto$

$1, 3 \mapsto 1, 5 \mapsto 0, 7 \mapsto 0, 9 \mapsto 1)\}$. Then we have $\Omega(F_2, \{p\}) = \Omega(F_2, \varnothing) = (1 \mapsto 1, 4 \mapsto 0, 6 \mapsto 0, 8 \mapsto 1)$. Therefore $\lceil \mathsf{Pre}_O(\downarrow F_2) \rceil \cap \mathsf{safe} = \lceil \mathsf{CPre}(\{F_1\}) \rceil \cap \mathsf{safe} = \{(1 \mapsto 1, 4 \mapsto 0, 6 \mapsto 0, 8 \mapsto 1)\}$. At the end of the computation, we get the fixpoint $\downarrow \{F := (1 \mapsto 1, 4 \mapsto -1, 6 \mapsto -1, 9 \mapsto 1)\}$. Since the initial state F_0 is in $\downarrow F$, Player O has a winning strategy and the formula is realizable. From the fixpoint computation, we can extract a winning strategy for the controller. An example is given by the Moore machine M depicted in Fig. 2. Note that for all $\sigma \in \Sigma_I$, $\mathsf{succ}(\mathsf{succ}(F, \{p\}), \sigma) \in \downarrow F$. By monotonicity of succ and the fact that $F \in \mathsf{safe}$, the runs on the words defined by this machine visit only non-final states in $\det(A, K)$, so that $L(M) \subseteq L_{\text{uc},0}(\det(A, K)) = L_{\text{uc},K}(A) \subseteq L_{\text{uc}}(A)$.

5 Performance Evaluation

In this section, we briefly present our implementation Acacia and compare it to Lily [9]. More information can be found online [2]. Acacia is a prototype implementation of our antichain algorithm for checking LTL realizability. To achieve a fair comparison, as Lily, Acacia is written in Perl. For a controller specification, which consists of an LTL formula and a partition $\{I, O\}$ of input/output variables, Acacia tests realizability of the specification. If it is realizable, it outputs a Moore machine representing a winning strategy for the controller, whose correctness can be verified by model-checking tools, otherwise it outputs a winning strategy for the environment. As Lily, Acacia runs in two steps. The first step builds a tbUCW for the formula, and the second step checks realizability of the automaton. We borrow the LTL-to-tbUCW construction procedure from Lily such that we can exclude the influence of automata construction to the performance comparison between Acacia and Lily[6]. We carried out experiments on a Linux platform with a 2.1GHz CPU and 2GB of memory. We compared Acacia and Lily on the test suite included in Lily, and on other examples derived from Lily's examples, as detailed further. As shown in the previous section (Theorem 5), realizability or unrealizability tests are slightly different, as we test unrealizability by testing the realizability by the environment of the negation of the specification. In the experiments, depending on whether the formula is realizable or not, we only report the results for the realizability or unrealizability tests. In practice, those two tests should be run in parallel. Tables 1 and 2 report the results of the tests for realizable and unrealizable examples respectively. In those tables,

Column tbUCW ***St./Tr.*** gives the number of states and transitions of the tbUCWs transformed from LTL formula. One may encounter $\varnothing$ when Lily's tbUCW optimization procedure concludes the language emptiness of the tbUCW.

Column tbUCW ***Time(s)*** gives the time (in seconds) spent in building up the tbUCWs. For realizability tests, Lily and Acacia construct the same tbUCW, while for unrealizability tests, they are different (as shown in Section 4, we use the tbUCW corresponding to the negation of the formula).

Column ***Rank*** gives the maximal rank used by Lily when trying to transform the tbUCW to an alternating weak tree automaton. *Rank* is a complexity measure for Lily.

[6] In Lily, this first step produces universal co-Büchi tree automata over Σ_O-labeled Σ_I-trees, which can easily be seen as tbUCWs over inputs Σ_I and outputs Σ_O. Although the two models are close, we introduced tbUCWs for the sake of clarity.

Column *Check Time(s)* gives the time (in seconds) spent in realizability checking. If the language of a tbUCW is proved to be empty during the optimization stage, Lily will directly conclude for unrealizability.
Column *K* reports the minimal K for which Acacia was able to conclude realizability of the tbUKCW. Usually, K is small for realizable specifications.
Column *No. Iter.* gives the number of iterations to compute the fixpoint.
Column $max\{|\mathsf{Pre}_I|\}/max\{|\mathsf{Pre}_O|\}$ reports the maximal sizes of the antichains obtained during the fixpoint computation when applying Pre_I and Pre_O respectively.

Lily's test suite includes examples 1 to 23. Except examples 1, 2, 4, and 11, they are all realizable. Table 1 shows, except demo 16, Acacia performs much better than Lily in realizability tests. For unrealizability tests, if we do not take into account the time for tbUCW construction, Acacia performs better as well.

In the test suite, demo 3 describes a scheduler. We have taken a scalability test by introducing more clients. In Table 1, from 3.4 to 3.6, when the number of clients reached 4, Lily ran over-time (> 3600 seconds). However, Acacia managed in finishing the check within the time bound.

One can weaken/strengthen a specification by removing/appending environment assumptions and controller assertions. We have carried out a diagnostic test based on demo 22. In the test cases from 22.3 to 22.9, the environment assumptions are getting stronger and stronger. The specifications turn out to be realizable after the case 22.5. A controller with a stronger environment shall be easier to realize. The data in Table 1, from 22.5 to 22.9, confirm our predication. For unrealizability check, in Table 2 from 22.1 to 22.4, both tools spent more time on case 22.4 than on case 22.3. However, Acacia turns out to be better for *Check Time*. Finally, we can see that the bottleneck for examples 22.1 to 22.9, as well as for examples 20 to 22, is the time spent to construct the automaton.
As a conclusion, the experiments show that the AntiChain algorithm can speed up the realizability check significantly and therefore show that it is a promising approach.

6 Summary

This paper described a novel Safraless approach to the LTL realizability problem, based on universal K-Co-Büchi word automata, which can be easily determinized. By exploiting the structure of the determinized automata, we were able to define an antichain algorithm to solve the realizability problem, which have been successfully implemented in the tool Acacia. In particular, an experimental evaluation showed that Acacia outperforms the existing tool Lily. These results come without any further optimizations. It raises a lot of questions for future works, we give only a few of them. As shown by the experimental results, the bottleneck of this approach is often the construction of the automaton from the formula, and the time spent on the realizability checking is highly related to the size of the automaton. Given a bound k, one could apply heuristics to prune the automaton, and in particular branches that visits accepting states more than k times. For a small k, this would reduce the size of the automaton drastically. On the other hand, the LTL-to-automata construction that we borrowed from Lily constructs an

			Lily		Acacia			
	tbUCW	tbUCW	Rank	Check	K	No.	$\max\{\lvert\mathsf{Pre}_O\rvert\}$/	Check
	St./Tr.	Time(s)	Time(s)	Time(s)	Iter.		$\max\{\lvert\mathsf{Pre}_I\rvert\}$	Time(s)
3	10/28	0.97	1	0.30	0	2	2 /2	0.00
5	13/47	1.53	1	0.65	0	2	2 /2	0.01
6	19/63	3.19	1	0.91	0	3	3 /3	0.03
7	11/34	1.42	1	0.31	0	2	2 /2	0.01
8	3/6	0.07	1	0.02	0	1	1 /1	0.00
9	5/10	0.33	1	0.03	1	6	3 /2	0.01
10	7/21	0.63	1	0.10	0	1	1 /1	0.00
12	8/26	0.35	1	0.07	0	1	1 /1	0.00
13	3/4	0.02	1	0.01	1	3	2 /1	0.00
14	5/13	0.26	1	0.07	1	4	3 /3	0.01
15	6/16	0.24	1	0.11	2	9	9 /13	0.08
16	8/26	0.57	1	1.45	3	16	64/104	7.89
17	6/20	0.40	1	0.31	2	12	8 /7	0.10
18	8/31	0.92	1	2.35	2	12	19/19	0.89
19	7/17	0.75	3	4.05	2	12	5/5	0.03
20	25/198	7.03	1	0.99	0	3	1 /1	0.04
21	13/72	15.61	1	1.88	0	4	25/13	0.40
22	19/115	25.28	1	1.21	1	7	4 /7	0.10
23	7/12	0.47	1	0.04	2	2	2 /1	0.00
3.1	10/28	1.09	1	0.31	1	2	2 / 2	0.01
3.2	18/80	2.60	1	7.70	1	2	4 / 4	0.07
3.3	26/200	2.60	1	554.99	1	2	8 / 8	0.65
3.4	34/480	7.59	-	> 3600	1	2	16/ 16	8.46
3.5	42/1128	12.46	-	> 3600	1	2	32/ 32	138.18
3.6	50/2608	22.76	-	> 3600	1	2	64/64	2080.63
22.5	7/38	4.17	1	0.50	2	19	4/6	0.12
22.6	19/115	21.20	1	1.52	1	7	4/7	0.11
22.7	13/75	7.51	1	0.73	1	6	3/4	0.05
22.8	10/50	3.82	1	0.43	1	5	2/3	0.03
22.9	7/29	1.46	1	0.33	1	5	2/3	0.02

Table 1. Performance comparison for realizability test

	Lily			Acacia					
	tbUCW	tbUCW	Check	tbUCW	tbUCW	K	No.	$\max\{\lvert\mathsf{Pre}_I\rvert\}$/	Check
	St./Tr.	Time(s)	Time(s)	St./Tr.	Time(s)		Iter.	$\max\{\lvert\mathsf{Pre}_O\rvert\}$	Time(s)
1	$\emptyset$	0.17	0.01	6/27	0.24	1	1	1/1	0.00
2	$\emptyset$	0.17	0.01	18/101	1.89	3	6	1/1	0.05
4	18/56	3.53	1.13	23/121	3.16	2	8	3/4	0.14
11	$\emptyset$	1.32	0.04	3/10	0.07	0	1	1/1	0.00
22.1	5/9	0.18	0.09	22/126	4.97	1	5	2/2	0.05
22.2	4/14	0.32	0.11	23/126	4.85	1	4	1/1	0.04
22.3	5/15	0.36	0.11	23/130	6.25	1	5	2/2	0.06
22.4	6/34	2.48	0.18	26/137	6.47	1	10	12/10	0.38

Table 2. Performance comparison for unrealizability test

automaton with symbolic transitions (labeled with Boolean formulas). Our implementation does not benefit from this compact representation, as it enumerates all the sets of labels that satisfy the Boolean formulas on the transitions. An optimization would be to handle those symbolic transitions efficiently. Another promising future work would be to combine our approach to the antichain algorithm for LTL satisfiability [20], which does not construct an automaton explicitly. Finally, on a more theoretical point of view, we would like to prove the correctness of our Safraless algorithm with a Safraless proof, as the proof of Lemma 2 still relies on the Safra's determinization procedure.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP*, 1989.
2. Acacia. Available at `http://www.antichains.be`, 2009.
3. R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *CONCUR*, pages 163–178. Springer, 1998.
4. A. Bouajjani, P. Habermehl, L. Holík, T. Touili, and T. Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In *CIAA*, pages 57–67, 2008.
5. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.
6. L. Doyen and J.-F. Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS*, volume 4424 of *LNCS*, pages 451–465. Springer, 2007.
7. S. Fogarty and M. Vardi. Buechi complementation and size-change termination. 2009. to appear in TACAS.
8. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
9. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124. IEEE Computer Society, 2006.
10. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.
11. D. Martin. Borel determinacy. In *Annals of Mathematics*, volume 102, pages 363–371, 1975.
12. Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
13. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages (POPL)*. ACM, 1989.
14. J.-F. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.
15. R. Rosner. *Modular synthesis of reactive systems*. Ph.d. dissertation, Weizmann Institute of Science, 1992.
16. Theo C. Ruys and Gerard J. Holzmann. Advanced spin tutorial. In *SPIN*, volume 2989 of *LNCS*, pages 304–305. Springer, 2004.
17. S. Safra. On the complexity of ω automata. In *FOCS*, pages 319–327, 1988.
18. F. Somenzi and R. Bloem. Efficient büchi automata from LTL formulae. In *CAV*, 2000.
19. W. Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science*, volume 4800 of *LNCS*, pages 635–655. Springer, 2008.
20. M. De Wulf, L. Doyen, N. Maquet, and J.-F. Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In *TACAS*, volume 4963 of *LNCS*, pages 63–77, 2008.

A Missing Proofs

A.1 Proof of Lemma 1

Proof. The back direction is obvious since $L_{\mathsf{uc},k}(A) \subseteq L_{\mathsf{uc}}(A)$ for all $k \in \mathbb{N}$.
For the forth direction, we first transform M into a tbNBW A_M such that $L(M) = L_{\mathsf{b}}(A_M)$. It suffices to copy every state of M and to define the transitions as follows: if $q \xrightarrow{i} q'$ is a transition of M with $i \in \Sigma_I$, and the output of q is $o \in \Sigma_O$, then we transform this transition into the two A_M transitions $q \xrightarrow{o} q^c$ and $q^c \xrightarrow{i} q'$ where q^c is a fresh state denoting a copy of q. All states of A_M are set to be final, so that $L_{\mathsf{b}}(A_M)$ is exactly the set of traces of infinite paths of A_M (viewed as an edge-labeled graph) starting from the initial state. By hypothesis, $L_{\mathsf{b}}(A_M) \subseteq L_{\mathsf{uc}}(A)$. Note that A_M has $2m$ states.
Let Q be the set of states of A, Q_{A_M} the set of states of A_M and $A \times A_M$ the product of A and A_M (i.e. the automaton over $\Sigma_I \cup \Sigma_O$ whose set of states is $Q \times Q_{A_M}$, initial states are pairs of initial states, and transitions have the form $(q,p) \xrightarrow{\sigma} (q',p')$ for all transitions $q \xrightarrow{\sigma} q'$ of A and $p \xrightarrow{\sigma} p'$ of A_M). Since $L_{\mathsf{b}}(A_M) \subseteq L_{\mathsf{uc}}(A)$, there is no cycle in $A \times A_M$ reachable from an initial state and that contains a state (q,p) where $q \in Q$ is final. Indeed, otherwise there would exist an infinite path in $A \times A_M$ visiting (q,p) infinitely often. Every infinite word obtained as a trace of this path would be accepted by A_M but not by A (since there would be a run on it visiting q infinitely often). Therefore the runs of A on words accepted by A_M visit at most $2nm$ final states, where n (resp. $2m$) is the number of states of A (resp. A_M). □

A.2 Proof of Lemma 2

Parity Conditions The proof uses the parity acceptance condition for automata and for games. Given an automaton B with state set Q_B, a parity acceptance condition is given by a mapping c from Q_B to $\mathbb{N}$. A run ρ is accepting if $min\{c(q) \mid q \in \mathsf{Inf}(\rho)\}$ is even. Final states are not needed in B for this acceptance condition. We denote by $L_{par,c}(B)$ the language accepted by B under the parity acceptance condition c.

Given a turn-based two-player game $G = (S_O, S_I, s_0, \Delta)$, the parity winning condition is given by a mapping $c : S_O \cup S_I \to \mathbb{N}$. In that case, for all $i \in \{O, I\}$, a strategy λ_i for Player i is winning if $\mathsf{Outcome}_G(\lambda_i) \subseteq \{\pi \in (S_O S_I)^\omega \mid min\{c(s) \mid s \in \mathsf{Inf}(\pi)\}$ is even$\}$.

Proof Let $A = (\Sigma_O, \Sigma_I, Q_O, Q_I, q_0, \alpha, \delta_O, \delta_I)$. We let $Q = Q_O \cup Q_I$ and $\delta = \delta_O \cup \delta_I$. Let A_{OI} be the automaton $(\Sigma, Q, q_0, \alpha, \delta)$. We denote by $m : (\Sigma_O \Sigma_I)^\omega \to \Sigma^\omega$ the function that maps any word $w = o_0 i_0 o_1 i_1 \dots$ to $m(w) = (o_0 \cup i_0)(o_1 \cup i_1) \dots$. Note that m admits an inverse denoted by m^{-1}. We have that $m(L_{\mathsf{b}}(A_{OI})) = L_{\mathsf{b}}(A)$ (*). By Safra's determinization, there exists a deterministic parity automaton D_{OI} with a parity condition c such that $L_{par,c}(D_{OI}) = L_{\mathsf{b}}(A_{OI})$. Moreover, by [12], we can assume that D_{OI} has at most n^{2n+2} states. Since $L_{par,c}(D_{OI}) \subseteq (\Sigma_I \Sigma_O)^\omega$, it is easy to transform D_{OI} into a deterministic turn-based parity automaton D with a parity condition c' such that $L_{par,c}(D_{OI}) = m^{-1}(L_{par,c'}(D))$: it suffices to take the product with the two-states automaton that accepts $(\Sigma_O \Sigma_I)^\omega$ (let i and o its two states). The states of the product are therefore pairs (q,p) with q a state of D_{OI} and $p \in \{i, o\}$, and we let $c'(q,p) = c(q)$. Note that D has at most $2n^{2n+2}$ states. From equality (*) and the equalities $L_{par,c}(D_{OI}) = m^{-1}(L_{par,c'}(D))$ and $L_{par,c}(D_{OI}) = L_{\mathsf{b}}(A_{OI})$, we get $L_{par,c'}(D) = L_{\mathsf{b}}(A)$. Then we complete the automaton D by adding two dead states and get a complete deterministic turn-based automaton A^d (with at most $2n^{2n+2}+2$ states). Finally, we take the dual parity condition $c_d = c'+1$ which increments the value of each state by 1, so that $L_{par,c_d}(A^d) = \Sigma^\omega - L_{par,c'}(D) = \Sigma^\omega - L_{\mathsf{b}}(A)$, from which we get $L_{\text{uc}}(A) = L_{par,c_d}(A^d)$.

Let $A^d = (\Sigma_O, \Sigma_I, Q^d_I, Q^d_O, q_0, \delta^d_I, \delta^d_O)$ and $Q^d = Q^d_I \cup Q^d_O$. We now view A^d has a turn-based two-player parity game $G(A^d) = (Q^d_0, Q^d_I, q_0, \Delta)$: Q^d_O are Player O's states (q_0 being the initial state) while Q^d_I are Player I's states, and we put a transition $(q,p) \in \Delta$ from a state $q \in Q^d$ to a state $p \in Q^d$ if there exists $\sigma \in \Sigma_I \cup \Sigma_O$ and a transition $q \xrightarrow{\sigma} p$ in A^d. Since A^d has at most $2n^{2n+2}+2$ states, $G(A^d)$ has also at most $2n^{2n+2}$ states.

The specification A^d is realizable (or equivalently A is realizable) iff Player O has a winning strategy in $G(A^d)$. Therefore if A is realizable, Player O has a winning strategy in $G(A^d)$ given by a mapping γ from Q^d_O to Q^d_I such that $\mathsf{Outcome}_{G(A^d)}(\gamma)$ are words ρ over $(Q^d_O Q^d_I)^\omega$ such that $min\{c(q) \mid q \in \mathsf{Inf}(\rho)\}$ is even. Moreover, those words correspond to accepting runs of A^d on words over Σ. Therefore the strategy γ can easily be used to define a Moore machine M such that $L(M) \subseteq L_{par,c}(A^d) = L_{\text{uc}}(A)$: first we assume that Σ is totally ordered. The machine M is defined as follows: Q^d_O are its states, q_0 is the initial state, the output function g is defined by $g(q) = min\{\sigma_o \mid (q, \sigma_o, \gamma(q)) \in \delta^d_O\}$, for all $q \in Q^d_O$, and finally we put a transition $q \xrightarrow{\sigma_i} q'$, for all $q, q' \in Q^d_O$, and all $\sigma_i \in \Sigma_I$ if $\gamma(q) \xrightarrow{\sigma_i} q' \in \delta^d_I$. Note that M is complete since A^d is complete, and has less than $(2n^{2n+2}+2)/2 = n^{2n+2}+1$ states. □

A.3 Proof of Theorem 2

Proof. Suppose that A is realizable. By Theorem 1, (A,K) is also realizable, as well as $\det(A,K)$. Thus there is exists non-empty and complete Moore machine M over inputs Σ_I and outputs Σ_O such that $L(M) \subseteq L_{\text{uc},0}(\det(A,K))$. We now construct a winning strategy γ for Player O in $G(A,K)$. Intuitively, $\mathsf{Outcome}_{G(A,K)}(\gamma)$ will correspond to runs of $\det(A,K)$ on words of $L(M)$. Therefore, since $L(M) \subseteq L_{\text{uc},0}(\det(A,K))$, $\mathsf{Outcome}_{G(A,K)}(\gamma)$ won't visit final states. For the sake of clarity, we view this Moore machine as a (total) mapping $\lambda : \Sigma_I^* \rightarrow \Sigma_O$. First assume that Σ_O and Σ_I are totally ordered by some order $\prec$. Take $H = F^O_0 F^I_0 \dots F^I_{m-1} F^O_m \in (\mathcal{F}_O \mathcal{F}_I)^* \mathcal{F}_O$ a finite play of length $2m+1$ in $G(A,K)$. The word H defines a word $w(H) \in \mathcal{F}_I^*$ as follows: $w(H) = \sigma^I_1 \dots \sigma^I_m$ where for all $1 \leq i \leq m$, $\sigma^I_i = min\{\sigma \mid \mathsf{succ}(F^I_{i-1}, \sigma) = F^O_i\}$. We let $\gamma(H) = \mathsf{succ}(F^O_m, \lambda(w(H)))$ (it exists since $\det(A,K)$ is complete, by Prop. 1).
We now prove that γ is winning. Let $H = F^O_0 F^I_0 F^O_1 F^I_1 \dots \in (\mathcal{F}_O \mathcal{F}_I)^\omega \cap \mathsf{Outcome}_{G(A,K)}(\gamma)$. For all $i \in \{0, \dots, n\}$, let $H_i = F^O_0 F^I_0 \dots F^O_i F^I_i$. We associate H with a word $u_H = (\sigma^O_0 \cup \sigma^I_0)(\sigma^O_1 \cup \sigma^I_1) \dots \in \Sigma_P^\omega$ where for all $i \geq 0$, $\sigma^O_i = \lambda(\sigma^I_0 \dots \sigma^I_{i-1})$ and $\sigma^I_i = min\{\sigma \mid \mathsf{succ}(F^I_i, \sigma) = F^O_{i+1}\}$. Note that since λ is winning, $u_H \in L_{\text{uc},0}(\det(A,K))$. Moreover, by definition of γ, for all $i \geq 0$, $\mathsf{succ}(F^O_i, \sigma^O_i) = F^I_i$. Therefore H is the run of $\det(A,K)$ on u_H and all states of H are not final states.

Conversely, suppose that Player O has a winning strategy γ in $G(A,K)$. It is known that we can assume that γ is memoryless [8]. So let γ be a mapping from $\mathcal{F}_O$ to $\mathcal{F}_I$.

We construct a winning strategy for the controller, represented as a Moore machine $M_\gamma = (\Sigma_O, \Sigma_I, \mathcal{F}_O, F_0, \delta_\gamma, g_\gamma)$. Its state set is $\mathcal{F}_O$ with initial state F_0; for all $F \in \mathcal{F}_O$, the output of F is defined by $g_\gamma(F) = \sigma$, for some $\sigma \in \Sigma_O$ such that $\delta_O(F, \sigma_o) = \gamma(F)$ (it exists since $(F, \gamma(F)) \in T$); for all $F \in \mathcal{F}_O$, and all $\sigma_i \in \Sigma_I$, the transition function is defined by $\delta_\gamma(F, \sigma_i) = \delta_I(\gamma(F), \sigma_i)$. Since γ is winning, it is clear by construction that all states of M_γ reachable from the initial state are non-final. Therefore $L(M_\gamma) \subseteq L_{\text{uc},0}(\det(A, K)) = L_{\text{uc}}(A)$. Moreover, M_γ is complete, since $\det(A, K)$ is complete. Since there is a winning strategy γ in $G(A, K)$, it means that $G(A, K)$ is non-empty, and so is M_γ, which concludes the proof. □

A.4 Proof of Proposition 3

Proof. (i) It holds as max$(-1, F(q) - q \in \alpha)$ $\leq$max$(-1, F'(q) - q \in \alpha)$, $\forall q \in Q$.

(ii) it holds as min$(K+1, F(q') + q \in \alpha)$ $\leq$min$(K+1, F'(q') + q \in \alpha)$, $\forall q' \in Q$.

(iii) Let $q \in Q$. We show that for all $q' \in \delta(q, \sigma)$, $F(q) \leq \mathsf{succ}(F, \sigma)(q') - (q' \in \alpha)$. This will be sufficient to conclude since it implies that $F(q) \leq max(-1, \mathsf{succ}(F, \sigma)(q') - (q' \in \alpha))$, for all $q' \in \delta(q, \sigma)$, and therefore that $F(q) \leq \Omega(\mathsf{succ}(F, \sigma), \sigma)(q)$.
So let $q' \in \delta(q, \sigma)$, and let $I(q') = \{q'' \mid (q'', \sigma, q') \in \delta, F(q'') \neq -1\}$. Since $(q, \sigma, q') \in \delta$, we have $q \in I(q')$. We know that $\mathsf{succ}(F, \sigma)(q') = max\{min(K + 1, F(q'') + q' \in \alpha) \mid q'' \in I(q')\}$. Since $q \in I(q')$, $\mathsf{succ}(F, \sigma)(q') \geq min(K + 1, F(q) + q' \in \alpha)$. If $F(q) + q' \in \alpha \leq K + 1$, then $\mathsf{succ}(F, \sigma)(q') - (q' \in \alpha) \geq F(q)$. The case $F(q) + (q' \in \alpha)$ is impossible since $F(q) \leq K$, as $F \in \mathsf{safe}$.

(iv) Let $q \in Q$. We first show that for all q' such that $(q', \sigma, q) \in \delta$ and $\Omega(F, \sigma)(q') \neq -1$, $\Omega(F, \sigma)(q') \leq F(q) - (q \in \alpha)$. This will be sufficient to conclude since it implies that $min(K + 1, \Omega(F, \sigma)(q') + (q \in \alpha)) \leq F(q)$, for all q' such that $(q', \sigma, q) \in \delta$, and therefore that $\mathsf{succ}(\Omega(F, \sigma), \sigma)(q) \leq F(q)$.
So let q' such that $(q', \sigma, q) \in \delta$ and $\Omega(F, \sigma)(q') \neq -1$. Let $I(q') = \{q'' \mid (q', \sigma, q'') \in \delta\}$. Since $(q', \sigma, q) \in \delta$, we have $q \in I(q')$. We know that $\Omega(F, \sigma)(q') = min\{max(-1, F(q'') - (q'' \in \alpha)) \mid q'' \in I(q')\}$. Since $q \in I(q')$, we get $\Omega(F, \sigma)(q') \leq max(-1, F(q) - (q \in \alpha))$. The case $F(q) - (q \in \alpha) < -1$ is impossible, since otherwise we would have $\Omega(F, \sigma)(q') = -1$, which contradicts the hypothesis. Therefore $max(-1, F(q) - (q \in \alpha)) = F(q) - (q \in \alpha)$ and $\Omega(F, \sigma)(q') \leq F(q) - (q \in \alpha)$. □

A.5 Proof of Proposition 5

Proof. We start by a simple remark. Let A be tbUKCW with input states Q_I and output states Q_O. When $K = 0$, Player I's locations in $G(A, K)$ are exactly the subsets of Q_I and Player O's locations are the subsets of Q_O, and the partial order $\preceq$ corresponds to set inclusion.

Now, let us consider for each $e = \{v, v'\} \in E$ the antichain (for $\subseteq$) $L_{\{v,v'\}} = \{V \setminus \{v\}, V \setminus \{v'\}\}$, L compactly represents all the subsets of V that are independent of the edge $\{v, v'\}$. Clearly $\mathsf{IND}(G) = \bigcap_{\{v,v'\} \in E} \downarrow L_{\{v,v'\}}$. As a direct consequence of the NP completeness of the independent set problem, it can not exist a polynomial time algorithm to compute the antichain for this intersection unless $P = NP$. Indeed, this antichain contains the maximal independent sets.

Now, we show how to construct a UKCW A and an antichain of subsets of states L such that $\mathsf{Pre}_I(\mathsf{Pre}_O(\mathsf{Pre}_O(L))$ is exactly $\mathsf{IND}(G)$. We can assume that V is totally ordered by some order, and for all edges $e = \{v, v'\} \in E$, we denote by $\pi_1(e)$ the minimal element of e and by $\pi_2(e)$ its maximal element. Now, the set of state of the automaton is structured in four layers: $S_3 = \{ok, ko\}$ belongs to Player I, $S_2 = \{(v, e, i) \mid v \in V, e \in E, i \in \{1, 2\}\}$ belongs to Player O, $S_1 = \{(v, e) \mid v \in V, e \in E\}$ belongs to Player O. Finally, $S_0 = \{v \mid v \in V\}$ belongs to Player I. Note that we do not make players strictly alternate here to simplify the exposition, it is easy to add a layer between the two actions of Player O to make the automaton turn-based. In those additional states, Player I would have only one action and the operation Pre_I would simulate the identity. The objective of Player O is to ensure that the control ends up in state $ok \in S_3$, so we take $L = \{\{ok\}\}$. Now, we explain how to put transitions between states and compute $\mathsf{Pre}_I(\mathsf{Pre}_O(\mathsf{Pre}_O(((L)))$. The transitions from S_2 and S_3 are $\{((v, e, i), (e, i), ok) \mid \pi_i(e) \neq v\} \cup \{((v, e', i), (e, i), ko) \mid \pi_i(e) = v \vee e \neq e')\}$, it is easy to verify that $\mathsf{Pre}_O(L)$ is equal to $\downarrow \cup_{e \in E}\{\{(v, e, 1) \mid v \neq \pi_1(e)\}, \{(v, e, 2) \mid v \neq \pi_2(e)\}$. The transitions from S_1 to S_2 are $\{((v, e), i, (v, e, i)) \mid i \in \{1, 2\} \wedge v \in V \wedge e \in E\}$. As states of S_1 belongs to Player O the controllable configurations $\mathsf{Pre}_O(\mathsf{Pre}_O((L))$ are $\downarrow \cup_{e \in E}\{\{(v, e) \mid v \neq \pi_1(e)\}, \{(v, e) \mid v \neq \pi_2(e)\}\}$. The transitions from S_0 to S_1 are $\{(v, e, (v, e)) \mid v \in V \wedge e \in E\}$. As states in S_0 belongs to Player I, we have that $\downarrow\mathsf{Pre}_I(\mathsf{Pre}_O(\mathsf{Pre}_O((L))) = \mathsf{IND}(G)$, indeed Player I can decide to verify any edge for independence, so only independent set of vertices can be in $\mathsf{Pre}_I(\mathsf{Pre}_O(\mathsf{Pre}_O((L)))$. □