An Antichain Algorithm for LTL Realizability[⋆]

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin

CS, Faculty of Sciences
Université Libre de Bruxelles (U.L.B.), Belgium

Abstract. In this paper, we study the structure of underlying automata based constructions for solving the LTL realizability and synthesis problem. We show how to reduce the LTL realizability problem to a game with an observer that 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 is 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 [4,5,6,14,21] 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 and synthesis. Although LTL realizability is 2EXPTIME-COMPLETE, 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 it.

The realizability problem for an LTL formula ϕ is best seen as a game between two players [13]. Each of the players 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"

[⋆] Work supported by the projects: (i) Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded Systems", http://www.quasimodo.aau.dk, (ii) Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/, and (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Government.

© Springer-Verlag Berlin Heidelberg 2009

(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 Player O wins if the resulting infinite word w is a model of ϕ. 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].[2]

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 of ϕ, transform A_ϕ into a deterministic Rabin automaton B using Safra's determinization procedure [18], 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. Indeed, 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. Moreover, 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[3].

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[4]. 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 this paper, we propose a different and more direct Safraless decision procedure for the LTL realizability and synthesis problem and we identify structural properties that allow us to define an antichain algorithm in the line of our previous works. We highlight differences with [10,9] in Section 5. 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

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

[2] Older works also consider the realizability problem but for more expressive and computationally intractable formalisms, see [20].

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

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

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 the problem of the realizability of a k-bounded UCW specification. Contrarily 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 in a tool called Acacia and our experiments show promising results. Indeed, even without further optimizations, Acacia outperforms Lily.

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.

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 and Synthesis.* As recalled in the introduction, the realizability problem for LTL is best seen as a game between two players. Each of the players 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 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 $\lambda_O : (\Sigma_O\Sigma_I)^* \to \Sigma_O$ while a strategy for Player I is a (total) mapping $\lambda_I : \Sigma_O(\Sigma_I\Sigma_O)^* \to \Sigma_I$. 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 $i_j = \lambda_I(o_0 i_0 \ldots o_{j-1} i_{j-1} o_j)$. 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*. If an LTL specification is realizable, there exists a finite-state strategy that realizes it [13]. The *synthesis problem* is to find a finite-state strategy that realizes the LTL specification.

E.g., let $I = \{q\}$, $O = \{p\}$ and $\psi = p\mathcal{U}q$. The formula ψ is not realizable. As q is controlled by the environment, he can decide to leave it always false and the outcome does not satisfy ϕ. However $\Diamond q \to (p\mathcal{U}q)$ 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 $\forall q \in Q \cdot \forall \sigma \in \Sigma \cdot |\delta(q, \sigma)| \leq 1$. It is *complete* if $\forall q \in Q \cdot \forall \sigma \in \Sigma \cdot \delta(q, \sigma) \neq \varnothing$. 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 $\forall i \geq 0 \cdot q_{i+1} \in \delta(q_i, \rho_i)$. 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.):

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

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 [19]) 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 with 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

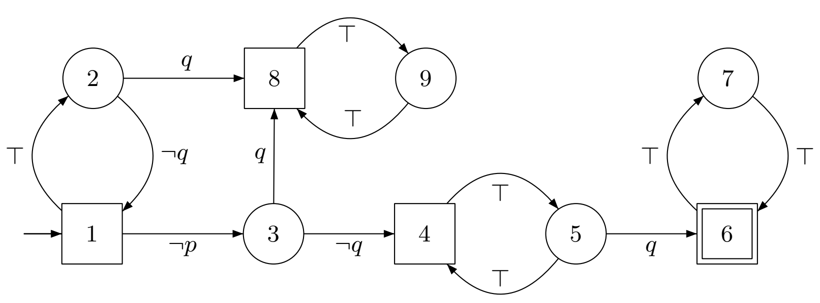

Fig. 1. tbUCW for $\Diamond q \rightarrow (p\mathcal{U}q)$ where $I = \{q\}$ and $O = \{p\}$

turn-based automata are always complete. Turn-based automata A still run on words 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. LTL realizability is equivalent to LTL realizability by a finite-state strategy [13]. We use Moore machines to represent finite-state strategies. 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 (total) transition function, and $g_M : Q \rightarrow \Sigma_O$ is a (total) output function. 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 $j \geq 0$, $\delta_M^*(i_0 \dots i_{j-1})$ is defined and $o_j = g_M(\delta_M^*(i_0 \dots i_{j-1}))$. In particular, $o_0 = g_M(\delta_M^*(\epsilon)) = g_M(q_0)$. The size of a Moore machine is defined similarly as the size of an automaton.

Thanks to previous remarks, the LTL realizability problem reduces to decide, given a tbUCW A over inputs Σ_I and outputs Σ_O, whether there is a non-empty Moore machine M such that $L(M) \subseteq L_{\text{uc}}(A)$. In this case we say that A is realizable. In our implementation, the tbUCW is equivalent to an LTL formula given as input and is constructed by using *Wring* [19].

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. Informally, the infinite paths of M starting from the initial state define 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. □

The following result is proved in Th. 4.3 of [10], as a small model property of universal co-Büchi tree automata. We also prove it here for the sake of self-containdness.

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

Proof. We sketch the proof. In the 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 Lem. 2, there is a non-empty Moore machine M with m states ($m \leq n^{2n+2}+1$) realizing A. Thus $L(M) \subseteq L_{\text{uc}}(A)$ and by Lem. 1, it 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 as $L_{\text{uc},K}(A) \subseteq L_{\text{uc}}(A)$. □

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 complete turn-based deterministic 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 play 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 $\pi = \pi_0\pi_1\pi_2 \cdots \in (S_O S_I)^\omega$ 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 tbUKCW $(\Sigma_O, \Sigma_I, Q_O, Q_I, q_0, \alpha, \Delta_O, \Delta_I)$ with $K \in \mathbb{N}$. We let $Q = Q_O \cup Q_I$ and $\Delta = \Delta_O \cup \Delta_I$. It is easy to construct an equivalent complete turn-based deterministic 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. We set the counter of a state q to -1 when 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. For any $n \in \mathbb{N}$, $[n]$ denotes the set $\{-1, 0, 1, \dots, n\}$. Formally, we let $\det(A, K) = (\Sigma_O, \Sigma_I, \mathcal{F}_O, \mathcal{F}_I, F_0, \alpha', \delta_O, \delta_I)$ where:

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

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 can be ignored since $\det(A, K)$ is deterministic. Formally, $G(A, K) = (\mathcal{F}_O, \mathcal{F}_I, F_0, T, \mathsf{safe})$ where $\mathsf{safe} = \mathcal{F} \backslash \alpha'$ and $T = \{(F, F') \mid \exists \sigma \in \Sigma_O \cup \Sigma_I, F' = \mathsf{succ}(F, \sigma)\}$. As an obvious consequence of Th. 1 and Prop. 1, we get:.

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 fixpoint 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 $2^{\mathcal{F}}$:

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

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

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 Th. 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 algorithm above 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 algorithms for union, intersection, inclusion and membership. Since the size of a state $F \in \mathcal{F}$ is in practice much smaller than the number of elements in the antichains, we consider that comparing two states is 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 $O(|L_1|)$.*

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 $\sigma \in \Sigma_I \cup \Sigma_O$,*

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

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 independent 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 in 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* U*K*CW *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 U*K*CWs: 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 realizable for Player O for a small value of K or $\neg\phi$ is realizable for Player I for a small value of K.

Synthesis. If a UCW A is realizable, it is easy to extract from the greatest fixpoint computation a Moore machine that realizes it. Let $\Pi_I \subseteq \mathcal{F}_I \cap \mathsf{safe}$ and $\Pi_O \subseteq \mathcal{F}_O \cap \mathsf{safe}$ be the two sets obtained by the greatest fixpoint computation. In particular, Π_I and Π_O are downward-closed and $\mathsf{Pre}_O(\Pi_I) = \Pi_O$, $\mathsf{Pre}_I(\Pi_O) = \Pi_I$. By definition of Pre_O, for all $F \in \lceil \Pi_O \rceil$, there exists $\sigma_F \in \Sigma$ such that $\mathsf{succ}(F, \sigma_F) \in \Pi_I$, and this σ_F can be computed. From this we can extract a Moore machine whose set of states is $\lceil \Pi_O \rceil$, the output function maps any state $F \in \lceil \Pi_O \rceil$ to σ_F, and the transition function, when reading some $\sigma \in \Sigma_I$, maps F to a state $F' \in \lceil \Pi_O \rceil$ such that $\mathsf{succ}(\mathsf{succ}(F, \sigma_F), \sigma) \preceq F'$ (it exists by definition of the fixpoint and by monotonicity of succ). The initial state is some state $F \in \lceil \Pi_O \rceil$ such that $F_0 \preceq F$ (it exists if the specification is realizable). Let M be this Moore machine. For any word w accepted by M, it is clear that w is also accepted by $\det(A, K)$, as succ is monotonic and $\Pi_O \subseteq \mathsf{safe}$. Therefore $L(M) \subseteq L_{\text{uc},0}(\det(A, K)) = L_{\text{uc},K}(A) \subseteq L_{\text{uc}}(A)$.

Example. We apply the antichain algorithm on the tbUCW depicted in Fig. 1, with $K = 1$. Remember 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 $\downarrow\{F := (1 \mapsto 1, 4 \mapsto -1, 6 \mapsto -1, 8 \mapsto 1)\}$. Since the initial state F_0 is in $\downarrow F$, Player O has a winning strategy and the formula is realizable. Fig. 2 shows a Moore machine obtained from the fixpoint computation.

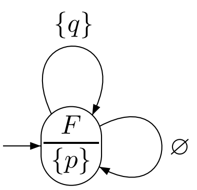

Fig. 2. Moore machine

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 LTL realizability and synthesis. To achieve a fair comparison, Acacia is written in Perl as Lily. Given an LTL formula and a partition of its propositions into inputs and outputs, Acacia tests realizability of the formula. If it is realizable, it outputs a Moore machine representing a winning strategy for the output player[5], otherwise it outputs a winning strategy for the input player. 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. As Lily, we borrow the LTL-to-tbUCW construction procedure from *Wring* [19] and adopt the automaton optimizations from Lily, so 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 in the sequel. As shown in the previous section (Th. 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.

Results. Tables 1 and 2 report on the results of the tests for unrealizable and realizable examples respectively. In those tables, **Column *formula size*** gives the size of the

[5] Note that the correctness of this Moore machine can be automatically verified by model-checking tools if desired.

[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 (as all our developments are done on construction for word automata).

Table 1. Performance comparison for unrealizability test

		Lily			Acacia					
	formula size	tbUCW St./Tr.	tbUCW Time(s)	Check Time(s)	tbUCW St./Tr.	tbUCW Time(s)	K	No. Iter.	max{|Pre$_I$|}/ max{|Pre$_O$|}	Check Time(s)
1	28	∅	0.17	0.01	6/27	0.24	1	1	1/1	0.00
2	28	∅	0.17	0.01	18/101	1.89	3	6	1/1	0.05
4	38	18/56	3.53	1.13	23/121	3.16	2	8	3/4	0.14
11	12	∅	1.32	0.04	3/10	0.07	0	1	1/1	0.00
22.1	24	5/9	0.18	0.09	22/126	4.97	1	5	2/2	0.05
22.2	23	4/14	0.32	0.11	23/126	4.85	1	4	1/1	0.04
22.3	29	5/15	0.36	0.11	23/130	6.25	1	5	2/2	0.06
22.4	37	6/34	2.48	0.18	26/137	6.47	1	10	12/10	0.38

formulas (number of atomic propositions and connectives). **Column** tbUCW ***St./Tr.*** gives the number of states and transitions of the tbUCWs transformed from LTL formula. One may encounter ∅ when Lily's tbUCW optimization procedure concludes the language emptiness of the tbUCW. **Column** tbUCW ***Time(s)*** gives the time (in seconds) spent on 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. **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 on the maximal sizes of the antichains obtained during the fixpoint computation when applying Pre_I and Pre_O respectively.

Comments. Lily's test suite includes examples 1 to 23. Except examples 1, 2, 4, and 11, they are all realizable. Table 2 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 2, 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 2, from 22.5 to 22.9, confirm this. For unrealizability check, in Table 1 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. With regards to this concern, the improvement in time complexity compared to Lily is less impressive. However, it was not expected that this first step of the algorithm (constructing the NBW for the LTL formula) would have been the bottleneck

Table 2. Performance comparison for realizability test

			Lily		Acacia				
	formula size	tbUCW St./Tr.	tbUCW Time(s)	Rank	Check Time(s)	K	No. Iter.	max{|Pre$_O$|}/ max{|Pre$_I$|}	Check Time(s)
3	34	10/28	0.97	1	0.30	0	2	2 /2	0.00
5	44	13/47	1.53	1	0.65	0	2	2 /2	0.01
6	49	19/63	3.19	1	0.91	0	3	3 /3	0.03
7	50	11/34	1.42	1	0.31	0	2	2 /2	0.01
8	7	3/6	0.07	1	0.02	0	1	1 /1	0.00
9	22	5/10	0.33	1	0.03	1	6	3 /2	0.01
10	13	7/21	0.63	1	0.10	0	1	1 /1	0.00
12	14	8/26	0.35	1	0.07	0	1	1 /1	0.00
13	11	3/4	0.02	1	0.01	1	3	2 /1	0.00
14	21	5/13	0.26	1	0.07	1	4	3 /3	0.01
15	31	6/16	0.24	1	0.11	2	9	9 /13	0.08
16	56	8/26	0.57	1	1.45	3	16	64/104	7.89
17	37	6/20	0.40	1	0.31	2	12	8 /7	0.10
18	63	8/31	0.92	1	2.35	2	12	19/19	0.89
19	32	7/17	0.75	3	4.05	2	12	5/5	0.03
20	72	25/198	7.03	1	0.99	0	3	1 /1	0.04
21	119	13/72	15.61	1	1.88	0	4	25/13	0.40
22	62	19/115	25.28	1	1.21	1	7	4 /7	0.10
23	19	7/12	0.47	1	0.04	2	2	2 /1	0.00
3.1	34	10/28	1.09	1	0.31	0	2	2 / 2	0.01
3.2	63	18/80	2.60	1	7.70	0	2	4 / 4	0.07
3.3	92	26/200	2.60	1	554.99	0	2	8 / 8	0.65
3.4	121	34/480	7.59	-	> 3600	0	2	16/ 16	8.46
3.5	150	42/1128	12.46	-	> 3600	0	2	32/ 32	138.18
3.6	179	50/2608	22.76	-	> 3600	1	2	64/64	2080.63
22.5	41	7/38	4.17	1	0.50	2	19	4/6	0.12
22.6	62	19/115	21.20	1	1.52	1	7	4/7	0.11
22.7	56	13/75	7.51	1	0.73	1	6	3/4	0.05
22.8	51	10/50	3.82	1	0.43	1	5	2/3	0.03
22.9	47	7/29	1.46	1	0.33	1	5	2/3	0.02

of the approach. Indeed, the problem is 2EXPTIME-COMPLETE, while the automata construction is in EXPTIME, and in [13], the forseen bottleneck is clearly the second step that relies on Safra's determinization.

As a conclusion, the experiments show that the antichain algorithm is a very promising approach to LTL synthesis. Although the formulas are still rather small, the results validate the relevance of the method. Indeed, without any further optimization, the results outperform Lily. We think that our algorithm is a step towards the realization of a tool that can handle specifications of practical interest.

Comparison with Kupferman-Vardi's Approach (implemented in Lily*).* In [10], the authors give a Safraless procedure for LTL synthesis. It is a three steps algorithm: (i) transform an LTL formula into a universal co-Büchi tree automaton (UCT) A that accepts the winning strategies of the controller, (ii) transform A into an alternating weak tree automaton B (AWT) such that $L(B) \neq \varnothing$ iff $L(A) \neq \varnothing$, (iii) transform B into an equivalent Büchi tree automaton C (NBT) and test its emptiness. This latter problem can be seen as solving a game with a Büchi objective. This approach differs from our approach in the following points. First, in [10], the author somehow reduce the realizability problem to a game with a *Büchi objective*, while our approach reduces it to a game with a *safety objective*. Second, our approach allows one to define a natural partial order on states that can be exploited by an antichain algorithm, which is not obvious in the approach of [10]. Finally, in [10], states of AWT are equipped with unique ranks that partition the set of states into layers. States which share the same rank are either all accepting or all non-accepting. The transition function allows one to stay in the same layer or to go in a layer with lower rank. A run is accepting if it gets stuck in a non-accepting layer. While our notion of counters looks similar to ranks, it is different. Indeed, the notion of rank does not constraint the runs to visit accepting states a bounded number of times (bounded by a constant). This is why a Büchi acceptance condition is needed, while counting the number of visited accepting states allows us to define a safety acceptance condition. However, we conjecture that when our approach concludes for realizability with bound k, the algorithm of [10] can conclude for realizability with a maximal rank linearly bounded by k. The converse is not true, we can define a class of examples where the maximal rank needed by [10] is 1 while our approach necessarily needs to visit at least an exponential number of accepting states. This is because the ranks does not count the number of accepting states, but counts somehow the number of finite sequences of accepting states of a certain type. We think that it is an interesting question for future research to see how the two methods can benefit from each other, and to formally prove the conjecture above.

6 Summary

This paper described a novel Safraless approach to LTL realizability and synthesis, based on universal K-Co-Büchi word automata. These automata can be easily determinized, and enjoy a structure that allowed us to define an antichain algorithm for LTL realizability, implemented in the tool Acacia. The results are very promising, as Acacia outperforms the existing tool Lily without any further optimizations (apart from antichains) while Lily uses clever optimizations to make the Vardi and Kupferman algorithm practical. Note that our approach also applies to any logic which can be translated into a UCW, and in particular, any logic closed by negation which can be translated into an NBW[7].

We plan to optimize Acacia in several ways. First, as the construction of the nondeterministic automaton from the LTL formula is currently the bottleneck of our approach, we would like to translate LTL formulas (in linear time) into alternating word automata, and then to UCW by applying the Miyano-Hayashi (MH) construction [17] implicitely as in [6,21]. The difficulty here is to find an adequate symbolic representation of counting functions for the implicit MH state space. Second, we would like to study how a compositional approach to realizability could apply to specifications which are large conjunctions of (small) sub-specifications.

Acknowledgments. We are grateful to the referees for their valuable comments and we warmly thank Laurent Doyen for his helpful remarks.

[7] Note also that any ω-regular specification can be expressed as a UCW, as a consequence our method is applicable to all such objective.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Ronchi Della Rocca, S. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. Acacia (2009), `http://www.antichains.be`
3. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
4. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: CIAA, pp. 57–67 (2008)
5. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)
6. Doyen, L., Raskin, J.-F.: Improved algorithms for the automata-based approach to model-checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 451–465. Springer, Heidelberg (2007)
7. Fogarty, S., Vardi, M.: Buechi complementation and size-change termination. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 16–30. Springer, Heidelberg (2009)
8. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
9. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: FMCAD, pp. 117–124. IEEE Computer Society, Los Alamitos (2006)
10. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS: IEEE Symposium on Foundations of Computer Science (FOCS) (2005)
11. Martin, D.: Borel determinacy. Annals of Mathematics 102, 363–371 (1975)
12. Piterman, N.: From nondeterministic büchi and streett automata to deterministic parity automata. Logical Methods in Computer Science 3(3) (2007)
13. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: ACM Symposium on Principles of Programming Languages (POPL). ACM, New York (1989)
14. Raskin, J.-F., Chatterjee, K., Doyen, L., Henzinger, T.A.: Algorithms for omega-regular games with imperfect information. Logical Methods in Computer Science 3(3) (2007)
15. Rosner, R.: Modular synthesis of reactive systems. Ph.d. dissertation, Weizmann Institute of Science (1992)
16. Ruys, T.C., Holzmann, G.J.: Advanced SPIN Tutorial. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 304–305. Springer, Heidelberg (2004)
17. Miyano, S., Hayashi, T.: Alternating automata on ω-words. Theoretical Computer Science 32, 321–330 (1984)
18. Safra, S.: On the complexity of ω automata. In: FOCS, pp. 319–327 (1988)
19. Somenzi, F., Bloem, R.: Efficient büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855. Springer, Heidelberg (2000)
20. Thomas, W.: Church's problem and a tour through automata theory. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 635–655. Springer, Heidelberg (2008)
21. De Wulf, M., Doyen, L., Maquet, N., Raskin, J.-F.: Antichains: Alternative algorithms for LTL satisfiability and model-checking. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 63–77. Springer, Heidelberg (2008)