Safraless Synthesis for Epistemic Temporal Specifications*

Rodica Bozianu[1], Cătălin Dima[1], and Emmanuel Filiot[2,**]

[1] Université Paris Est, LACL (EA 4219), UPEC, 94010 Créteil Cedex, France
[2] Université Libre de Bruxelles, CP 212 - 1050 Bruxelles Belgium

Abstract. In this paper we address the synthesis problem for specifications given in linear temporal single-agent epistemic logic, KLTL (or KL_1), over single-agent systems having imperfect information of the environment state. [17] have shown that this problem is 2Exptime complete. However, their procedure relies on complex automata constructions that are notoriously resistant to efficient implementations as they use Safra-like determinization.

We propose a "Safraless" synthesis procedure for a large fragment of KLTL. The construction transforms first the synthesis problem into the problem of checking emptiness for universal co-Büchi tree automata using an information-set construction. Then we build a safety game that can be solved using an antichain-based symbolic technique exploiting the structure of the underlying automata. The technique is implemented and applied to a couple of case studies.

1 Introduction

The goal of system verification is to check that a system satisfies a given property. One of the major achievements in system verification is the theory of *model checking*, that uses automata-based techniques to check properties expressed in temporal logics, for systems modelled as transitions systems. The *synthesis* problem is more ambitious: given a specification of the system, the aim is to automatically synthesise a system that fulfils the constraints defined by the specification. Therefore, the constraints do not need to be checked a posteriori, and this allows the designer to focus on defining high-level specifications, rather than designing complex computational models of the systems.

Reactive systems are non-terminating systems that interact with some environment, e.g., hardware or software that control transportations systems, or medical devices. One of the main challenge of synthesis of reactive systems is to cope with the uncontrollable behaviour of the environment, which usually leads to computationally harder decision problems, compared to system verification. For instance, model-checking properties expressed in *linear time temporal logic* (LTL) is PSpace-c while LTL synthesis is 2Exptime-c [14]. Synthesis of reactive systems from temporal specifications has gain a lot of interest recently as several works have shown its practical feasibility [13,4,3,12,9]. These progresses were supported by Kupferman and Vardi's breakthrough in automata-based synthesis techniques [13]. More precisely, they have shown that the complex Safra's determinization operation, used in the classical LTL synthesis algorithm [14], could be avoided by working directly with universal co-Büchi automata. Since then,

* Work partially supported by the ANR research project "EQINOCS" no. ANR-11-BS02-0004
** F.R.S.-FNRS research associate (chercheur qualifié)

several other "Safraless" procedures have been defined [13,16,10,9]. In [16,9], it is shown that LTL synthesis reduces to testing the emptiness of a universal co-Büchi tree automaton, that in turn can be reduced to solving a safety game. The structure of the safety games can be exploited to define a symbolic game solving algorithm based on compact antichain representations [9].

In these works, the system is assumed to have perfect information about the state of the environment. However in many practical scenarios, this assumption is not realistic since some environment information may be hidden to the system (e.g. private variables). Towards the (more realistic) synthesis of partially informed systems, imperfect information two-player games on graphs have been studied [15,7,2,8]. However, they consider explicit state transition systems rather than synthesis from temporal specifications. Moreover, the winning objectives that they consider cannot express fine properties about imperfect information, i.e., cannot speak about knowledge.

Epistemic Temporal Logics [11] are logics formatted for reasoning about multi-agent situations. They are extensions of temporal logics with knowledge operators K_i for each agent. They have been successfully used for verification of various distributed systems in which the knowledge of the agents is essential for the correctness of the system specification.

Synthesis problem with temporal epistemic objectives Vardi and van der Meyden [17] have considered epistemic temporal logics to define specifications that can, in addition to temporal properties, also express properties that refer to the imperfect information, and they studied the synthesis problem. They define the synthesis problem in a multi-agent setting, for specifications written in LTL extended with knowledge operators K_i for each agent (KLTL). In such models, transitions between states of the environment model depend on actions of the environment and the system. The system does not see which actions are played by the environment but get some observation on the states in which the environment can be (observations are subsets of states). An execution of the environment model, from the point of view of the system, is therefore an infinite sequence alternating between its own actions and observations.

The goal of the KLTL synthesis problem is to automatically generate a strategy for the system (if it exists) that tells it which action should be played, depending on finite histories, so that whatever the environment does, all the (concrete) infinite executions resulting from this strategy satisfy the KLTL formula. In [17], this problem was shown to be undecidable even for two agents against an environment. For a single agent, they show that the problem is 2Exptime-c, by reduction to the emptiness of alternating Büchi automata. This theoretically elegant construction is however difficult to implement and optimize, as it relies on complex Safra-like automata operations (Muller-Schupp construction).

Contributions In this paper, we follow the formalisation of [17] and, as our main contribution, define and implement a Safraless synthesis procedure for the positive fragment of KLTL (KLTL$^+$), i.e., KLTL formulas where the operator K does not occur under any negations. Our procedure relies on universal co-Büchi tree automata (UCT). More precisely, given a KLTL$^+$ formula φ and some environment model $\mathcal{M}_E$, we show how to construct a UCT $\mathcal{T}_\varphi$ whose language is exactly the set of strategies that realize φ in $\mathcal{M}_E$.

Despite the fact that our procedure has 2-ExpTime worst-case complexity, we have implemented it and shown its practical feasibility through a set of examples. In particular, based on ideas of [9], we reduce the problem of checking the emptiness of $\mathcal{T}_\varphi$ to solving a safety game whose state space can be ordered and compactly represented by antichains. Moreover, rather than using the reduction of [9] as a blackbox, we further optimize the antichain representations to improve their compactness. Our implementation is based on the tool Acacia [5] and, to the best of our knowledge, it is the first implementation of a synthesis procedure for epistemic temporal specifications. As an application, this implementation can be used to solve two-player games of imperfect information whose objectives are given as LTL formulas, or universal co-Büchi automata.

Organization of the paper In Section 2, we define the KLTL synthesis problem. In Section 3, we define universal co-Büchi automata for infinite words and trees. In Section 4, we consider the particular case of LTL synthesis in an environment model with imperfect information. The construction explained in that section will be used in the generalization to KLTL$^+$and moreover, it can be used to solve two-player imperfect information games with LTL (and more generally ω-regular) objectives. In Section 5, we define our Safraless procedure for KLTL$^+$, and show in Section 6 how to implement it with antichain-based symbolic techniques. Finally, we describe our implementation in Section 7. *Full proofs can be found in the full version of the paper[6] in which, for self-containdness, we also explain the reduction to safety games.*

2 KLTL Realizability and Synthesis

In this section, we define the realizability and synthesis problems for KLTL specifications, for one partially informed agent, called the *system*, against some environment.

Environment Model We assume to have, as input of the problem, a model of the behaviour of the environment as a transition system. This transition system is defined over two disjoint sets of actions Σ_1 and Σ_2, for the system and the environment respectively. The transition relation from states to states is defined with respect to pairs of actions in $\Sigma_1 \times \Sigma_2$. Additionally, each state s of the environment model carries an interpretation $\tau_e(s)$ over a (finite) set of propositions $\mathcal{P}$. However, the system is not perfectly informed about the value of some propositions, i.e., some propositions are visible to the system, and some are not. Therefore, we partition the set $\mathcal{P}$ into two sets $\mathcal{P}_v$ (the visible propositions) and $\mathcal{P}_i$ (the invisible ones).

An *environment model* is a tuple $\mathcal{M}_E = (\mathcal{P}, \Sigma_1, \Sigma_2, S_e, S_0, \Delta_e, \tau_e)$ where

- $\mathcal{P}$ is a finite set of propositions, Σ_1 and Σ_2 are finite set of actions for the system and the environment resp.,
- S_e is a set of states, $S_0 \subseteq S_e$ a set of initial states,
- $\tau_e : S_e \to 2^{\mathcal{P}}$ is a labelling function,
- $\Delta_e \subseteq S_e \times \Sigma_1 \times \Sigma_2 \times S_e$ is a transition relation.

The model is assumed to be deadlock-free, i.e. from any state, there exists at least one outgoing transition. Moreover, the model is assumed to be complete for all actions of the system, i.e. for all states and all actions of the system, there exists an outgoing transition. The set of *executions* of $\mathcal{M}_E$, denoted by exec($\mathcal{M}_E$), is the set of infinite sequences of states $\rho = s_0 s_1 \cdots \in S_e^\omega$ such that $s_0 \in S_0$ and for all

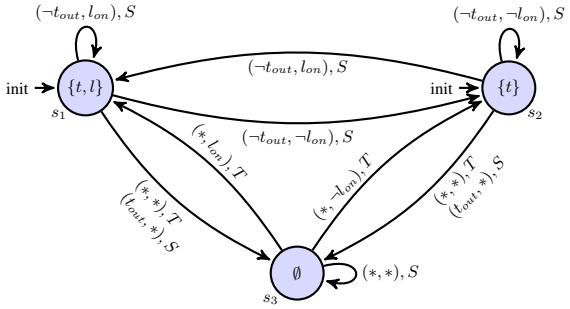

Fig. 1: Environment model $\mathcal{M}_E$ of Example 1

$i > 0$, $(s_i, a_1, a_2, s_{i+1}) \in \Delta_e$ for some $(a_1, a_2) \in \Sigma_1 \times \Sigma_2$. Given a sequence of states $\rho = s_0 s_1 \dots$ and a set $P \subseteq \mathcal{P}$, we denote by $\text{trace}_P(\rho)$ its projection over P, i.e. $\text{trace}_P(\rho) = (\tau_e(s_0) \cap P)(\tau_e(s_1) \cap P) \dots$. The *visible trace* of ρ is defined by $\text{trace}_v(\rho) = \text{trace}_{\mathcal{P}_v}(\rho)$. The *language* of $\mathcal{M}_E$ with respect to P is defined as $\mathcal{L}_P(\mathcal{M}_E) = \{\text{trace}_P(\rho) \mid \rho \in \text{exec}(\mathcal{M}_E)\}$. The language of $\mathcal{M}_E$ is defined as $\mathcal{L}_{\mathcal{P}}(\mathcal{M}_E)$. The visible language of $\mathcal{M}_E$ is defined as $\mathcal{L}_{\mathcal{P}_v}(\mathcal{M}_E)$. Finally, given an infinite sequence of actions $a = a_1^0 a_2^0 \dots \in (\Sigma_1.\Sigma_2)^\omega$ and an execution $\rho = s_0 s_1 \dots$ of $\mathcal{M}_E$, we say that a is compatible with ρ if for all $i \geq 0$, $(s_i, a_1^i, a_2^i, s_{i+1}) \in \Delta_e$.

This formalization is very close to that of [17]. However in [17], partial observation is modeled as a partition of the state space. The two models are equivalent. In particular, we will see that partitioning the propositions into visible and invisible ones also induces a partition of the state space into observations.

Example 1. We illustrate the notion of environment model on the example of [17], that describes the behaviour of an environment against a system acting on a timed toggle switch with two positions (on,off) and a light. It is depicted in Fig. 1. The set $\mathcal{P} = \{t, l\}$ contains two propositions t (true iff the toggle is on) and l (true iff the light is on). Actions of the system are $\Sigma_1 = \{T, S\}$ for "toggle" and "skip" respectively. The system can change the position of the toggle only if it plays T, and S has no effect. Actions of the environment are $\Sigma_2 = \{(t_{out}, l_{on}) \mid t_{out}, l_{on} \in \{0, 1\}\}$. The boolean variables t_{out} and l_{on} indicate that the environment times out the toggle and that it switches on the light. The transition function is depicted on the figure as well as the labelling function $\tau_e : S_e \to 2^{\mathcal{P}}$. The star $*$ means "any action". The light can be on only if the toggle in on (state s_1), but it can be off even if the toggle is on (state s_2), in case it is broken. This parameter is uncontrollable by the system, and therefore it is controlled by the environment (action l_{on}). The timer is assumed to be unreliable and therefore the environment can timeout at any time (action t_{out}). The system sees only the light, i.e. $\mathcal{P}_v = \{l\}$ and $\mathcal{P}_i = \{t\}$. The goal of the system is to have a strategy such that he always knows the position of the toggle.

Observations. The partition of the set of propositions $\mathcal{P}$ into a set of visible propositions $\mathcal{P}_v$ and a set of invisible propositions $\mathcal{P}_i$ induces an indistinguishability relation over the states S_e. Two states are *indistinguishable*, denoted $s_1 \sim s_2$, if they have the same visible propositions, i.e. $\tau_e(s_1) \cap \mathcal{P}_v = \tau_e(s_2) \cap \mathcal{P}_v$. It is easy to see that $\sim$ is an equivalence relation over S_e. Each equivalence class of S_e induced by $\sim$ is called

an *observation*. The equivalence class of a state $s \in S_e$ is denoted by $\mathrm{o}(s)$ and the set of observations is denoted by $\mathcal{O}$. The relation $\sim$ is naturally extended to (finite or infinite) executions: $\rho_1 \sim \rho_2$ if $\mathrm{trace}_v(\rho_1) = \mathrm{trace}_v(\rho_2)$. Similarly, two executions $\rho = s_0 s_1 \dots$ and $\rho' = s'_0 s'_1 \dots$ are said to be indistinguishable *up to some position* i if $\mathrm{trace}_v(s_0 \dots s_i) = \mathrm{trace}_v(s'_0 \dots s'_i)$. This indistinguishability notion is also an equivalence relation over executions that we denote by $\sim_i$.

Coming back to Example 1, since the set of visible propositions is $\mathcal{P}_v = \{l\}$ and the set of invisible ones is $\mathcal{P}_i = \{t\}$, the states s_2 and s_3 are indistinguishable (in both s_2, s_3 the light is off) and therefore $\mathcal{O} = \{o_0, o_1\}$ with $o_0 = \{s_2, s_3\}$ and $o_1 = \{s_1\}$.

Given an infinite sequence $u = a_1 o_1 a_2 o_2 \cdots \in (\Sigma_1.\mathcal{O})^\omega$ of actions of Player 1, and observations, we associate with u the set of possible executions of $\mathcal{M}_E$ that are compatible with u. Formally, we define $\mathrm{exec}(\mathcal{M}_E, u)$ the set of executions $\rho = s_0 s_1 \cdots \in \mathrm{exec}(\mathcal{M}_E)$ such that for all $i \geq 1$, $\mathrm{o}(s_i) = o_i$ and there exists an action b_i of the environment such that $(s_{i-1}, a_i, b_i, s_i) \in \Delta_e$. We also define the traces of u as the set of traces of all executions of $\mathcal{M}_E$ compatible with u, i.e. $\mathrm{traces}(u) = \{\mathrm{traces}(\rho) \mid \rho \in \mathrm{exec}(\mathcal{M}_E, u)\}$.

Epistemic Linear Time Temporal Logic (KLTL) We now define the logic KLTL for one-agent (the system). The logic KLTL extends the logic LTL with an epistemic operator $K\phi$, modelling the property that the system knows that the formula ϕ holds. *KLTL formulae* are defined over the set of atomic propositions $\mathcal{P}$ by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid K\varphi$$

in which $p \in \mathcal{P}$ and $\bigcirc$ and $\mathcal{U}$ are the "next" and "until" operators from linear temporal logic. Formulas of the type $K\varphi$ are read as "the system knows that φ holds". We define the macros $\Diamond$ (eventually) and $\Box$ (always) as usual. LTL is the fragment of KLTL without the K operator.

The semantics of a KLTL formula φ is defined for an environment model $\mathcal{M}_E = (\mathcal{P}, \Sigma_1, \Sigma_2, S_e, S_0, \Delta_e, \tau_e)$, a set of executions $R \subseteq \mathrm{exec}(\mathcal{M}_E)$, an execution $\rho = s_0 s_1 \cdots \in R$ and a position $i \geq 0$ in ρ. It is defined inductively:

- $R, \rho, i \models p$ if $p \in \tau_e(s_i)$,
- $R, \rho, i \models \neg\varphi$ if $R, \rho, i \not\models \varphi$,
- $R, \rho, i \models \varphi_1 \vee \varphi_2$ if $R, \rho, i \models \varphi_1$ or $R, \rho, i \models \varphi_2$,
- $R, \rho, i \models \bigcirc\varphi$ if $R, \rho, i+1 \models \varphi$,
- $R, \rho, i \models \varphi_1 \mathcal{U} \varphi_2$ if $\exists j \geq i$ s.t. $R, \rho, j \models \varphi_2$ and $\forall i \leq k < j$, $R, \rho, k \models \varphi_1$,
- $R, \rho, i \models K\varphi$ if for all $\rho' \in R$ s.t. $\rho \sim_i \rho'$, we have $R, \rho', i \models \varphi$.

In particular, the system knows φ at position i in the execution ρ, if all other executions in R whose prefix up to position i are indistinguishable from that of ρ, also satisfy φ. We write $R, \rho \models \varphi$ if $R, \rho, 0 \models \varphi$, and $R \models \varphi$ if $R, \rho \models \varphi$ for all executions $\rho \in R$. We also write $\mathcal{M}_E \models \varphi$ to mean $\mathrm{exec}(\mathcal{M}_E) \models \varphi$. Note that $\mathcal{M}_E \models \varphi$ iff $\mathcal{M}_E \models K\varphi$.

Consider Example 1 and the set R of executions that eventually loop in s_1. Pick any ρ in R. Then $R, \rho, 0 \models \Box K \Diamond (l)$. Indeed, take any position i in ρ and any other executions $\rho' \in R$ such that $\rho \sim_i \rho'$. Then since ρ' will eventually loop in s_1, it will satisfy $\Diamond(l)$. Therefore $R, \rho, i \models K\Diamond(l)$, for all $i \geq 0$.

KLTL Realizability and Synthesis As presented in [9] for the perfect information setting, the realizability problem, given the environment model $\mathcal{M}_E$ and the KLTL formula φ, is best seen as a turn-based game between the system (Player 1) and the environment (Player 2). In the first round of the play, Player 1 picks some action $a_1^0 \in \Sigma_1$, then Player 2 picks some action in $a_2^0 \in \Sigma_2$ and solves the nondeterminism in Δ_e, and a new round starts. The two players play for an infinite duration and the outcome is an infinite sequence $w = a_1^0 a_2^0 a_1^1 a_2^1 \ldots$. The winning objective is given by some KLTL formula φ. Player 1 wins the play if for all executions ρ of $\mathcal{M}_E$ that are compatible with w, we have $\mathcal{M}_E, \rho \models \varphi$.

Player 1 plays according to *strategies* (called *protocols* in [17]). Since Player 1 has only partial information about the state of the environment, his strategies are based on the histories of his own actions and the observations he got from the environment. Formally, a strategy for Player 1 is a mapping $\lambda : (\Sigma_1 \mathcal{O})^* \to \Sigma_1$, where, as defined before, $\mathcal{O}$ denotes the set of observations of Player 1 over the states of $\mathcal{M}_E$. Fixing a strategy λ of Player 1 restricts the set of executions of the environment model $\mathcal{M}_E$. An execution $\rho = s_0 s_1 \cdots \in \text{exec}(\mathcal{M}_E)$ is said to be *compatible* with λ if there exists an infinite sequence of actions $a = a_1^0 a_2^0 ... \in (\Sigma_1.\Sigma_2)^\omega$, compatible with ρ, such that for all $i \geq 0$, $a_1^i = \lambda(a_1^0 \text{o}(s_0) a_1^1 \text{o}(s_1) ... a_1^{i-1} \text{o}(s_{i-1}))$. We denote by $\text{exec}(\mathcal{M}_E, \lambda)$ the set of executions of $\mathcal{M}_E$ compatible with λ.

Definition 1. *A KLTL formula φ is realizable in $\mathcal{M}_E$ if there exists a strategy λ for the system such that* $\text{exec}(\mathcal{M}_E, \lambda) \models \phi$.

Theorem 1 (R. van der Meyden and M. Vardi in [17]). *The KLTL realizability problem (for one agent) is 2ExpTime-complete.*

If a formula is realizable, the *synthesis* problem asks to generate a finite-memory strategy that realizes the formula. Such a strategy always exists if the specification is realizable [17]. Finite memory strategies can be represented by Moore machines that read observations and output actions of Player 1. We refer the reader to [9] for a formal definition of finite-memory strategies.

Considering again Example 1, the formula $\varphi = \Box(K(t) \vee K(\neg t))$ expresses the fact that the system knows at each step the position of the toggle. As argued in [17], this formula is realizable if the initial set of the environment is $\{s_1, s_2\}$ since both states are labelled with t. Then, a winning strategy of the system is to play first time T(it will lead to s_3) and then always play S in order to stay in that state. Following this strategy, in the first step the formula $K(t)$ is satisfied and then $K(\neg t)$ becomes true forever. However, the formula is not realizable if the set of initial states of the environment is $\{s_2, s_3\}$ since from the beginning the system doesn't know the value of the toggle.

3 Automata for Infinite Words and Trees

Automata on Infinite Words An *infinite word automaton* over some (finite) alphabet Σ is a tuple $A = (\Sigma, Q, Q_0, \Delta, \alpha)$ where Σ is the finite input alphabet, Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\alpha \subseteq Q$ is the set of final states (accepting states) and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation.

For all $q \in Q$ and all $\sigma \in \Sigma$, we let $\Delta(q, \sigma) = \{q' | (q, \sigma, q') \in \Delta\}$. We let $|A| = |Q| + |\Delta|$. We say that A is *deterministic* if $|Q_0| = 1$ and $\forall q \in Q, \forall \sigma \in$

$\Sigma, |\Delta(q,\sigma)| \leq 1$. It is *complete* if $\forall q \in Q, \forall \sigma \in \Sigma, \Delta(q,\sigma) \neq \emptyset$. In this paper we assume, w.l.o.g., that the word automata are always complete.

A *run* of the automaton A on an infinite input word $w = w_0 w_1 w_2 ...$, is a sequence $r = q_0 q_1 q_2 ... \in Q^\omega$ such that $(q_i, w_i, q_{i+1}) \in \Delta$ for all $i \geq 0$ and $q_0 \in Q_0$. We denote by $Runs_A(w)$ the set of runs of A on w and by $Visit(r,q)$ the number of times the state q is visited along the run r(or ∞ if the path visit the state q infinitely often). Here, we consider two accepting conditions for infinite word automata and name the infinite word automata according to the accepting condition being used. Let $B \in \mathbb{N}$. A word $w \in \Sigma^\omega$ is accepted by A if (according to the accepting condition):

$$\text{Universal Co-Büchi} : \forall r \in Runs_A(w), \forall q \in \alpha,\ Visit(r,q) < \infty$$
$$\text{Universal B-Co-Büchi} : \forall r \in Runs_A(w), \forall q \in \alpha,\ Visit(r,q) \leq B$$

The set of words accepted by A with the universal co-Büchi (resp. B-co-Büchi) accepting condition is denoted by $\mathcal{L}_{uc}(A)$ (resp. $\mathcal{L}_{uc,B}(A)$) . We say that A is a *universal co-Büchi word automaton* (UCW) if the first acceptance condition is used and that (A, B) is an *universal B-co-Büchi word automaton* (UBCW) if the second one is used.

Given an LTL formula φ, we can translate it into an equivalent universal co-Büchi word automaton A_φ . This can be done with a single exponential blow-up by first negating φ, then translating $\neg\varphi$ into an equivalent nondeterministic Büchi word automaton, and then dualize it into a universal co-Büchi word automaton [9,13].

Automata on Infinite Trees Given a finite set D of directions, a $D{-}tree$ is a prefix-closed set $T \subseteq D^*$, i.e., if $x \cdot d \in T$, where $d \in D$, then $x \in T$. The elements of T are called *nodes* and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot d$, for $d \in D$, are the *successors* of x. A node x is a *leaf* if it has no successor in T, i.e., $\forall d \in D, x \cdot d \notin T$. The tree T is *complete* if for all nodes, there are successors in all directions, formally, $\forall x \in T, \forall d \in D, x \cdot d \in T$. Finite and infinite branches π in a tree T are naturally defined, respectively, as finite and infinite paths in T starting from the root node. Given an alphabet Σ, a *$\Sigma{-}$labelled $D{-}$tree* is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. We omit τ when it is clear from the context. Then, in a tree T, an infinite (resp. finite) branch π induces an infinite (resp. finite) sequence of labels and directions in $(\Sigma.D)^\omega$ (resp. $(\Sigma.D)^*\Sigma$). We denote this sequence by $\tau(\pi)$. For instance, for a set of system's actions Σ_1 and a set of observations $\mathcal{O}$, a strategy $\lambda : (\Sigma_1\mathcal{O})^* \to \Sigma_1$ of the system can be seen has a Σ_1-labelled $\mathcal{O}$-tree whose nodes are finite outcomes[3].

A *universal co-Büchi tree automaton* (UCT) is a tuple $\mathcal{T} = (\Sigma, Q, Q_0, D, \Delta, \alpha)$ where Σ is the finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, D is the set of directions, $\Delta : Q \times \Sigma \times D \to 2^Q$ is the transition relation (assumed to be total) and α is the set of final states. If the tree automaton is in some state q at some node x labelled by some $\sigma \in \Sigma$, it will evaluate, for all $d \in D$, the subtree rooted at $x.d$ in parallel from all the states of $\Delta(q, \sigma, d)$. Let us define the notion of run formally.

[3] Technically, a strategy λ is defined also for histories that are not accessible by λ itself from the initial (empty) history ϵ. The tree represents only accessible histories but we can, in the rest of the paper, assume that strategies are only defined for their accessible histories. Formally, we assume that a strategy is a partial function whose domain H satisfies $\epsilon \in H$ and for all $h \in H$ and all $o \in \mathcal{O}$, $h.\lambda(h).o \in H$, and H is minimal (for inclusion) w.r.t. this property.

For all $q \in Q$ and $\sigma \in \Sigma$, we denote by $\Delta(q, \sigma) = \{(q_1, d_1), \ldots, (q_n, d_n)\}$ the *disjoint* union of all sets $\Delta(q, \sigma, d)$ for all $d \in D$. A *run* of $\mathcal{T}$ on an infinite $\Sigma-labelled\ D-$tree $\langle T, \tau \rangle$ is a $(Q \times D^*)-labelled\ \mathbb{N}-tree\ \langle T_r, \tau_r \rangle$ such that $\tau_r(\epsilon) \in Q_0 \times \{\epsilon\}$ and, for all $x \in T_r$ such that $\tau_r(x) = (q, v)$, if $\Delta(q, \tau(v)) = \{(q_1, d_1), \ldots, (q_n, d_n)\}$, we have $x \cdot i \in T_r$ and $\tau_r(x \cdot i) = (q_i, v \cdot d_i)$ for all $0 < i \leq n$, . Note that there is at most one run per input tree (up to tree isomorphism). A run $\langle T_r, \tau_r \rangle$ is *accepting* if for all infinite branches π of T_r, $\tau_r(\pi)$ visits a finite number of accepting states. The language of $\mathcal{T}$, denoted by $\mathcal{L}_{uc}(\mathcal{T})$, is the set of Σ-labelled D-trees such that there exists an accepting run on them. Similarly, we define universal B-co-Büchi tree automata by strengthening the acceptance conditions on all branches to the B-co-Büchi condition.

As noted in [9,16], testing the emptiness of a UCT automaton reduces to testing the emptiness of a universal B-co-Büchi accepting condition for a sufficiently large bound B, which in turn reduces to solving a safety game. Symbolic techniques, that are also exploited in this paper, have been used to solve the safety games[9].

4 LTL synthesis under imperfect information

In this section, we first explain an automata-based procedure to decide realizability under imperfect information of LTL formulas against an environment model. This procedure will be extended in the next section to handle the K operator.

Take an environment model $\mathcal{M}_E = (\mathcal{P}, \Sigma_1, \Sigma_2, S_e, S_0, \Delta_e, \tau_e)$. Then, a complete Σ_1-labelled $\mathcal{O}$-tree $\langle T, \tau \rangle$ defines a strategy of the system. Any infinite branch π of $\langle T, \tau \rangle$ defines an infinite sequence of actions and observations of $\mathcal{M}_E$, which in turn corresponds to a set of possible traces in $\mathcal{M}_E$. We denote by traces(π) this set of traces, and it is formally defined by traces(π) = traces$(\tau(\pi))$ (recall that the set of traces of a sequence of actions and observations has been defined in Section 2).

Given an LTL formula ψ, we construct a universal co-Büchi tree automaton $\mathcal{T} = (\Sigma_1, Q, Q_0, \mathcal{O}, \Delta, \alpha)$ that accepts all the strategies of Player 1 (the system) that realize ψ under the environment model $\mathcal{M}_E$. First, one converts ψ into an equivalent UCW $\mathcal{A} = (2^{\mathcal{P}}, Q^{\mathcal{A}}, Q_0^{\mathcal{A}}, \Delta^{\mathcal{A}}, \alpha^{\mathcal{A}})$. Then, as a direct consequence of the definition of KLTL realizability:

Proposition 1. *Given a complete Σ_1-labelled $\mathcal{O}$-trees $\langle T, \tau \rangle$, $\langle T, \tau \rangle$ defines a strategy that realizes ψ under $\mathcal{M}_E$ iff for all infinite branches π of $\langle T, \tau \rangle$ and all traces $\rho \in$ traces(π), $\rho \in L(\mathcal{A})$.*

We now show how to construct a universal tree automaton that checks the property mentioned in the previous proposition, for all branches of the trees. We use universal transitions to check, on every branch of the tree, that all the possible traces (possibly uncountably many) compatible with the sequence of actions in Σ_1 and observations in $\mathcal{O}$ defined by the branch satisfy ψ. Based on finite sequences of observations that the system has received, it can define its *knowledge* I of the possible states in which the environment can be, as a subset of states of S_e. Given an action $a_1 \in \Sigma_1$ of the system and some observation $o \in \mathcal{O}$, we denote by $\text{post}_a(I, o)$ the new knowledge that the system can infer from observation o, action a and its previous information I. Formally, $\text{post}_a(I, o) = \{s \in S_e \cap o \mid \exists a_2 \in \Sigma_2, \exists s' \in I \text{ s.t. } (s', a, a_2, s) \in \Delta_e\}$.

The states of the universal tree automaton $\mathcal{T}$ are pairs of states of $\mathcal{A}$ and knowledges, plus some extra state $(q_w, \varnothing)$, i.e. $Q = Q^{\mathcal{A}} \times 2^{S_e} \cup \{(q_w, \varnothing)\}$ where $(q_w, \varnothing)$ is added

for completeness. The final states are defined as $\alpha = \alpha^{\mathcal{A}} \times 2^{S_e}$ and initial states as $Q_0 = Q_0^{\mathcal{A}} \times S_0$. To define the transition relation, let us consider a state $q \in Q^{\mathcal{A}}$, a knowledge set $I \subseteq S_e$, an action $a \in \Sigma_1$ and some observation $o \in \mathcal{O}$. We now define $\Delta((q, I), a, o)$. It could be the case that there is no transition in $\mathcal{M}_E$ from a state of I to a state of o, i.e. $\text{post}_a(I, o) = \varnothing$. In that case, all the paths from the next o-node of the tree should be accepting. This situation is modelled by going to the extra state $(q_w, \varnothing)$, i.e. $\Delta((q, I), a, o) = (q_w, \varnothing)$.

Now suppose that $\text{post}_a(I, o)$ is non-empty. Since the automaton must check that all the traces of $\mathcal{M}_E$ that are compatible with actions of Σ_1 and observations are accepted by $\mathcal{A}$, intuitively, one would define $\Delta((q, I), a, o)$ as the set of states of the form $(q', \text{post}_a(I, o))$ for all states q' such that there exists $s \in I$ such that $(q, \tau_e(s), q') \in \Delta^{\mathcal{A}}$. However, it is not correct for several reasons. First, it could be that s has no successor in o for action a, and therefore one should not consider it because the traces up to state s die at the next step after getting observation o. Therefore, one should only consider states of I that have a successor in o. Second, it is not correct to associate the new knowledge $\text{post}_a(I, o)$ with q', because it could be that there exists a state $s' \in \text{post}_a(I, o)$ such that for all its predecessors s'' in I, there is no transition $(q, \tau_e(s''), q')$ in $\Delta^{\mathcal{A}}$, and therefore, one would also take into account sequences of interpretations of propositions that do not correspond to any trace of $\mathcal{M}_E$.

Taking into account these two remarks, we define, for all states q', the set $I_{q,q'} = \{s \in I \mid (q, \tau(s), q') \in \Delta^{\mathcal{A}}\}$. Then, $\Delta((q, I), a, o)$ is defined as the set

$$\Delta((q, I), a, o) = \{(q', \text{post}_a(I_{q,q'}, o)) \mid \exists s \in I,\ (q, \tau(s), q') \in \Delta^{\mathcal{A}}\}$$

Note that, since $\bigcup_{q' \in Q^{\mathcal{A}}} \text{post}_a(I_{q,q'}, o) = \text{post}_a(I, o)$ and the automaton is universal, the system does not have better knowledge by restricting the knowledge sets.

Lemma 1. *The LTL formula ψ is realizable in $\mathcal{M}_E$ iff $\mathcal{L}(\mathcal{T}) \neq \varnothing$.*

Moreover, it is known that if a UCT has a non-empty language, then it accepts a tree that is the unfolding of a finite graph, or equivalently, that can be represented by a Moore machine. Therefore if ψ is realizable, it is realizable by a finite-memory strategy. In this paper we will also use the notation $\mathcal{T}_{\psi,X}$ for the UCT built for the LTL formula ψ where the executions of $\mathcal{M}_E$ start from the set $X \subseteq S_e$,i.e., $Q_0 = Q_0^{\mathcal{A}} \times X$.

5 Safraless procedure for positive KLTL synthesis

In this section, we extend the construction of Section 4 to the positive fragment of KLTL. Positive formulas are defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid K\varphi \mid \varphi \mathcal{U} \varphi$$

Note that this fragment is equivalent to the fragment of KLTL in which all the knowledge operators K in formulas occur under an even number of negations. This is obtained by straightforwardly pushing the negations downwards the atoms. We denote this fragment of KLTL by KLTL$^+$.

Sketch of the construction Given a KLTL$^+$ formula φ and an environment model $\mathcal{M}_E = (\mathcal{P}, \Sigma_1, \Sigma_2, S_e, S_0, \Delta_e, \tau_e)$, we show how to construct a UCT $\mathcal{T}$ such that $\mathcal{L}(\mathcal{T}) \neq \varnothing$ iff φ is realizable in $\mathcal{M}_E$. The construction is compositional and follows, for the basic blocks, the construction of Section 4 for LTL formulas. The main idea is to replace subformulas of the form $K\gamma$ by fresh atomic propositions k_γ so that we get an LTL formula for which the realizability problem can be transformed into the emptiness of a UCT. The realizability of the subformulas $K\gamma$ that have been replaced by k_γ is checked by branching universally to a UCT for γ, constructed as in Section 4. Since transitions are universal, this will ensure that all the infinite branches of the tree from the current node where a new UCT has been triggered also satisfy γ. The UCTs we construct are defined over an extended alphabet that contains the new atomic propositions, but we show that we can safely project the final UCT on the alphabet Σ_1. The assumption on positivity of KLTL formulas implies that there is no subformulas of the form $\neg K\gamma$. The rewriting of subformulas by fresh atomic propositions cannot be done in any order. We now describe it formally.

We inductively define a sequence of formulas associated with φ as: $\varphi^0 = \varphi$ and, for all $i > 0$, φ^i is the formula φ^{i-1} in which the *innermost* subformulas $K\gamma$ are replaced by fresh atomic propositions k_γ. Let d be the smallest index such that φ^d is an LTL formula (in other words, d is the maximal nesting level of K operators). Let $\mathbb{K}$ denote the set of new atomic propositions, i.e., $\mathbb{K} = \bigcup_{i=0}^{d} \{k_\gamma \mid K\gamma \in \varphi^i\}$, and let $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$. Note that by definition of the formulas φ^i, for all atomic proposition k_γ occurring in φ^i, γ is an LTL formula over $\mathcal{P}'$. E.g. if $\varphi = p \rightarrow K(q \rightarrow Kr \vee Kz)$ and $\mathcal{P} = \{p, q, r, z\}$, then the sequence of formulas is: $\varphi^0 = \varphi$, $\varphi^1 = p \rightarrow K(q \rightarrow k_r \vee k_z)$ $\varphi^2 = p \rightarrow k_\gamma$ where $\gamma = q \rightarrow k_r \vee k_z$.

Then, we construct incrementally a chain of universal co-Büchi tree automata $\mathcal{T}^d, \ldots, \mathcal{T}^0$ such that $\mathcal{L}(\mathcal{T}^d) \supseteq \mathcal{L}(\mathcal{T}^{d-1}) \supseteq \cdots \supseteq \mathcal{L}(\mathcal{T}^0)$ and, the following invariant is satisfied: for all $i \in \{0, \ldots, d\}$, $\mathcal{T}^i$ accepts exactly the set of strategies that realize φ^i in $\mathcal{M}_E$. Intuitively, the automaton $\mathcal{T}^i$ is defined by adding new transitions in $\mathcal{T}^{i+1}$, such that for all atomic propositions k_γ occurring in φ^{i+1}, $\mathcal{T}^i$ will ensure that $K\gamma$ is indeed satisfied, by branching to a UCT checking γ whenever the atomic proposition k_γ is met. Since formulas φ^i are defined over the extended alphabet $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$ and $\mathcal{M}_E$ is defined over $\mathcal{P}$, we now make clear what we mean by realizability of a formula φ^i in $\mathcal{M}_E$. It uses the notion of *extended model executions* and *extended strategies*.

Extended actions, model executions and strategies We extend the actions of the system to $\Sigma_1' = \Sigma_1 \times 2^{\mathbb{K}}$ (call e-actions). Informally, the system plays an e-action (a, K) if it considers formulas $K\gamma$ for all $k_\gamma \in K$ to be true. An *extended execution* (e-execution) of $\mathcal{M}_E$ is an infinite sequence $\rho = (s_0, K_0)... \in (S_e \times 2^{\mathbb{K}})^\omega$ such that $s_0 s_1 ... \in \text{exec}(\mathcal{M}_E)$. We denote $s_0 s_1 \ldots$ by $\text{proj}_1(\rho)$ and $K_0 K_1 \ldots$ by $\text{proj}_2(\rho)$. The extended labelling function τ_e' is a function from $S_e \times 2^{\mathbb{K}}$ to $\mathcal{P}'$ defined by $\tau_e'(s, K) = \tau_e(s) \cup K$. The indistinguishability relation between extended executions is defined, for any two extended executions ρ_1, ρ_2, by $\rho_1 \sim \rho_2$ iff $\text{proj}_1(\rho_1) \sim \text{proj}_1(\rho_2)$ and $\text{proj}_2(\rho_1) = \text{proj}_2(\rho_2)$, i.e., the propositions in $\mathbb{K}$ are visible to the system. We define $\sim_i$ over extended executions similarly. Given the extended labelling functions and indistinguishability relation, the KLTL satisfiability notion $R, \rho, i \models \psi$ can be naturally defined for a set of e-execution R, $\rho \in R$ and ψ a KLTL formula over $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$.

An extended strategy is a strategy defined over e-actions, i.e. a function from $(\Sigma_1'\mathcal{O})^*$ to Σ_1'. For an infinite sequence $u = (a_0, K_0)o_0(a_1, K_1)o_1 \cdots \in (\Sigma_1'\mathcal{O})^\omega$, we define $\text{proj}_1(u)$ as $a_0o_0\ldots$. The sequence u defines a set of compatible e-executions $\text{exec}(\mathcal{M}_E, u)$ as follows: it is the set of e-executions $\rho = (s_0, K_0)(s_1, K_1)... \in (S_e \times 2^{\mathbb{K}})^\omega$ such that $\text{proj}_1(\rho) \in \text{exec}(\mathcal{M}_E, \text{proj}_1(u))$. Similarly, we define for e-strategies λ' the set $\text{exec}(\mathcal{M}_E, \lambda')$ of e-executions compatible with λ'. A KLTL formula ψ over $\mathcal{P}'$ is realizable in $\mathcal{M}_E$ if there exists an e-strategy λ' such that for all runs $\rho \in \text{exec}(\mathcal{M}_E, \lambda')$, we have $\text{exec}(\mathcal{M}_E, \lambda'), \rho, 0 \models \psi$.

Proposition 2. *There exists an e-strategy $\lambda' : (\Sigma_1'\mathcal{O})^* \to \Sigma_1'$ realizing φ^0 in $\mathcal{M}_E$ iff there exists a strategy $\lambda : (\Sigma_1\mathcal{O})^* \to \Sigma_1$ realizing φ^0 in $\mathcal{M}_E$.*

Proof. Let see e-strategies and strategies as $\Sigma_1'-$labelled (resp. Σ_1-labelled) $\mathcal{O}$-trees. Given a tree representing λ', we project its labels on Σ_1 to get a tree representing λ. The strategy λ defined in this way realises φ^0, as φ^0 does not contain any occurrence of propositions in $\mathbb{K}$. Conversely, given a tree representing λ, we extend its labels with $\emptyset$ to get a tree representing λ'. It can be shown for the same reasons that λ' realizes φ^0. □

Incremental tree automata construction The invariant mentioned before can now be stated more precisely: for all i, $\mathcal{T}^i$ accepts the e-strategies $\lambda' : (\Sigma_1'\mathcal{O}) \to \Sigma_1'$ that realise φ^i in $\mathcal{M}_E$. Therefore, the UCT $\mathcal{T}^i$ are labelled with e-actions Σ_1'. We now explain how they are constructed.

Since φ^d is an LTL formula, we follow the construction of Section 4 to build the UCT $\mathcal{T}^d$. Then, we construct $\mathcal{T}^i$ from $\mathcal{T}^{i+1}$, for $0 \leq i < d$. The invariant tells us that $\mathcal{T}^{i+1}$ defines all the e-strategies that realize φ^{i+1} in $\mathcal{M}_E$. It is only an over-approximation of the set of e-strategies that realize φ^i in $\mathcal{M}_E$ (and *a fortiori* φ^0), since the subformulas of φ^i of the form $K\gamma$ correspond to atomic propositions k_γ in φ^{i+1}, and therefore $\mathcal{T}^{i+1}$ does not check that they are satisfied. Therefore to maintain the invariant, $\mathcal{T}^i$ is obtained from $\mathcal{T}^{i+1}$ such that whenever an action that contains some formula $k_\gamma \in sub(\varphi^{i+1})$ occurs on a transition of $\mathcal{T}^{i+1}$, we trigger (universally) a new transition to a UCT $\mathcal{T}_{\gamma,I}$, for the current information set I in $\mathcal{T}^{i+1}$, that will check that $K\gamma$ indeed holds. The assumption on positivity of KLTL formulas is necessary here as we do not have to check for formulas of the form $\neg K\gamma$, which could not be done without an involved "non Safraless" complementation step. Since γ is necessarily an LTL formula over $\mathcal{P}'$ by definition of the formula φ^{i+1}, we can apply the construction of Section 4 to build $\mathcal{T}_{\gamma,I}$.

Formally, from the incremental way of constructing the automata $\mathcal{T}^j$ for $j \geq i$, we know that $\mathcal{T}^{i+1}$ has a set of states Q_{i+1} where all states are of the form (q, I) where $I \subseteq S_e$ is some knowledge. In particular, it can be verified to be true for the state space of $\mathcal{T}^d$ by definition of the construction of Section 4. Let also Δ_{i+1} be the transition relation of $\mathcal{T}^{i+1}$. For all formulas γ such that k_γ occurs in φ^{i+1}, we let Q_γ be the set of states of $\mathcal{T}_{\gamma,I}$ and Δ_γ its set of transitions. Again from the construction of Section 4, we know that $Q_\gamma = Q \times 2^{S_e}$ where Q is the set of states of a UCW associated with γ (assumed to be disjoint from that of $\mathcal{T}^{i+1}$) and $Q^0_\gamma = Q \times I$.

We define the set of states Q_i of $\mathcal{T}^i$ by $Q_{i+1} \cup \bigcup_{k_\gamma \in sub(\varphi^{i+1})} Q_\gamma$. Its set of transitions Δ_i is defined as follows. Assume w.l.o.g. that there is a unique initial state $q_0 \in Q$ in the UCW $\mathcal{A}_\gamma$. If $(q', I') \in \Delta_{i+1}((q, I), (a, K), o)$ where $I, I' \subseteq S_e$,

$a \in \Sigma_1$, $K \subseteq \mathbb{K}$, $o \in \mathcal{O}$ and $k_\gamma \in K$ is such that $k\gamma$ occurs in φ^{i+1}, then we let $(q', I') \in \Delta_i((q, I), (a, K), o)$ and $\Delta_\gamma((q_0, I), (a, K), o) \subseteq \Delta_i((q, I), (a, K), o)$. The whole construction is given in [6], as well as the proof of its correctness. The invariant is satisfied:

Lemma 2. *For all $i \geq 0$, $\mathcal{L}(\mathcal{T}^i)$ accepts the set of e-strategies that realize φ^i in $\mathcal{M}_E$.*

From Lemma 2, we know that $\mathcal{L}(\mathcal{T}^0)$ accepts the set of e-strategies that realize $\varphi^0 = \varphi$ in $\mathcal{M}_E$. Then by Proposition 2 we get:

Corollary 1. *The* KLTL$^+$ *formula φ is realizable in $\mathcal{M}_E$ iff $\mathcal{L}(\mathcal{T}^0) \neq \varnothing$.*

We now let $\mathcal{T}_\varphi$ be the UCT obtained by projecting $\mathcal{T}^0$ on Σ_1. We have:

Theorem 2. *For any* KLTL$^+$ *formula φ, one can construct a UCT $\mathcal{T}_\varphi$ such that $\mathcal{L}(\mathcal{T}_\varphi)$ is the set of strategies that realize φ in $\mathcal{M}_E$.*

The number of states of $\mathcal{T}_\varphi$ is (in the worst-case) $2^{|S_e|}.(2^{|\varphi^d|} + \sum_{k_\gamma \in \mathbb{K}} 2^{|\gamma|})$, and since $|\varphi^d| + \sum_{\gamma \in \mathbb{K}} |\gamma|$ is bounded by $|\varphi|$, the number of states of $\mathcal{T}_\varphi$ is $O(2^{|S_e|+|\varphi|})$.

6 Antichain Algorithm

In the previous sections, we have shown how to reduce the problem of checking the realizability of a KLTL$^+$ formula φ to the emptiness of a UCT $\mathcal{T}_\varphi$ (Theorem 2). In this section, we describe an antichain symbolic algorithm to test the emptiness of $\mathcal{T}_\varphi$.

It is already known from [9] that checking emptiness of the language defined by a UCT $\mathcal{T}$ can be reduced to checking the emptiness of $\mathcal{L}_{uc,B}(\mathcal{T})$ for a sufficiently large bound B, which in turn can be reduced to solving a safety game. Clearly, for all $b \geq 0$, if $\mathcal{L}_{uc,b}(\mathcal{T}) \neq \emptyset$, then $\mathcal{L}_{uc}(\mathcal{T}) \neq \emptyset$. This has led to an incremental algorithm that starts with the bound 0, and the experiments have shown that in general, a small bound b is necessary to conclude for realizability of an LTL formula (transformed into the emptiness of a UCT). We also exploit this idea in our implementation and show that for the KLTL$^+$ specifications that we considered, this observation still holds: small bounds are enough.

In [9], it is shown that the safety games can be solve on-the-fly without constructing them explicitly, and that the fixpoint algorithm used to solve these safety games could be optimized by using some antichain representation of the sets constructed during the fixpoint computation. Rather than using the algorithm of [9] as a black box, we study the state space of the safety games constructed from the UCT $\mathcal{T}_\varphi$ and show that they are also equipped with a partial order that allows one to get more compact antichain representations. We briefly recall the reduction of [9], the full construction of the safety games is given in [6].

Given a bound $b \geq 0$ and a UCT $\mathcal{T}$, the idea is to construct a safety game $G(\mathcal{T}, b)$ such that Player 1 has a winning strategy in $G(\mathcal{T}, b)$ iff $L_{uc,b}(\mathcal{T})$ is non-empty. The game $G(\mathcal{T}, b)$ is obtained by extending the classical automata subset construction with counters which count, up to b, the maximal number of times all the runs, up to the current point, have visited accepting states. If Q is the set of states of $\mathcal{T}$, the set of states of the safety game $G(\mathcal{T}, b)$ is all the functions $F : Q \to \{-1, 0, \ldots, b+1\}$. The value

$F(q) = -1$ means that no run have reached q and $F(q) \in \{0, \ldots, b\}$ means that the maximal number of accepting states that has been visited by some run reaching q is $F(q)$. The safe states are all the functions F such that $F(q) \leq b$ for all $q \in Q$. The set of states can be partially ordered by the pairwise comparison between functions and it is shown that the sets of states manipulated by the fixpoint algorithm are downward closed for this order.

Consider now the UCT $\mathcal{T}_\varphi$ constructed from the KLTL$^+$ formula φ. Its state space is of the form $Q \times 2^{S_e}$ where S_e is the set of states of the environment, because the construction also takes into account the knowledge the system has from the environment. Given a bound b, the state space of the safety game $G(\mathcal{T}_\varphi, b)$ is therefore functions from $Q \times 2^{S_e}$ to $\{-1, \ldots, b+1\}$. However, we can reduce this state space thanks to the following result:

Proposition 3. *For all runs $\langle T_r, \tau \rangle$ of $\mathcal{T}_\varphi$ on some tree T, for all branches π, π' in T_r of the same length such that they follow the same sequence of observations, if $\tau(\pi) = (q, I)$ and $\tau(\pi') = (q, I')$, then $I = I'$.*

In other words, given the same sequence of observations, the tree automaton $\mathcal{T}_\varphi$ computes, for a given state q, the same knowledge.

Based on this proposition, it is clear that reachable states F of $G(\mathcal{T}_\varphi, b)$ satisfy, for all states $q \in Q$ and knowledges I, I', if $F(q, I) \neq -1$ and $F(q, I') \neq -1$ then $I = I'$. We can therefore define the state space of $G(\mathcal{T}_\varphi, b)$ as the set of pairs $(F, \overline{K})$ such that $F : Q \to \{-1, \ldots, b+1\}$ and $\overline{K} : Q \to 2^{S_e}$ associates with each state q a knowledge (we let $G(q) = \emptyset$ if $F(q) = -1$). This state space is naturally ordered by $(F_1, \overline{K}_1) \preceq (F_2, \overline{K}_2)$ if for all $q \in Q$, $F_1(q) \leq F_2(q)$ and $\overline{K}_1(q) \subseteq \overline{K}_2(q)$. We show that all the sets manipulated during the fixpoint computation used to solve the safety games are downward closed for this order and therefore can be represented by the antichain of their maximal elements. A detailed analysis of the size of the safety game shows that $G(\mathcal{T}_\phi, B)$ is doubly exponential in the size of φ, and therefore, since safety games can be solved in linear time, one gets a 2Exptime upper bound for KLTL$^+$ realizability. The technical details are given in [6].

7 Implementation and Case Studies

In this section we briefly present our prototype implementation Acacia-K for KLTL$^+$synthesis [1], and provide some interesting examples on which we tested the tool, on a laptop equipped with an Intel Core i7 2.10Ghz CPU. Acacia-K extends the LTL synthesis tool Acacia+[5]. As Acacia+, the implementation is made in Python together with C for the low level operations that need efficiency.

As Acacia+, the tool is available in one version working on both Linux and MacOsX and can be executed using the command-line interface. As parameters, in addition to the files containing the KLTL$^+$ formula and the partition of the signals and actions, Acacia-K requires a file with the environment model. The output of the tool is a winning strategy, if the formula is realizable, given as a Moore machine described in Verilog and if this strategy is small, Acacia-K also outputs it as a picture.

In order to have a more efficient implementation, the construction of the automata for the LTL formulas γ is made on demand. That is, we construct the UCT $\mathcal{T}_\gamma$ incrementally by updating it as soon as it needs to be triggered from some state (q, I) which has not been constructed yet.

As said before, the synthesis problem is reduced to the problem of solving a safety game for some bound b on the number of visits to accepting states. The tool is incremental: it tests realizability for small values of b first and increments it as long as it cannot conclude for realizability. In practice, we have observed, as for classical LTL synthesis, that small bounds b are sufficient to conclude for realizability. However if the formula is not realizable, we have to iterate up to a large upper bound, which in practice is too large to give an efficient procedure for testing unrealizability. We leave as future work the implementation of an efficient procedure for testing unrealizability.

Taking now Example 1, the strategy provided by the tool is depicted in Figure 2. It asks to play first "toggle" and then keep on playing "skip" and, depending on the observation he gets, the system goes in a different state. The state 0 is for the start, the state 1 is the "error" state in which the system goes if he receives a wrong observation. That is, the environment gives an observation even if he cannot go in a state having that observation. Then, if the observation is correct, after playing the action "toggle" from the initial states $\{s_1, s_2\}$, the environment is forced to go in s_3 and by playing the action "skip", the system forces the environment to stay in s_3 and he will know that t is false. In the strategy, this situation corresponds to the state 2. For this example, Acacia-K constructed a UCT with 31 states and the total running time is 0.2s.

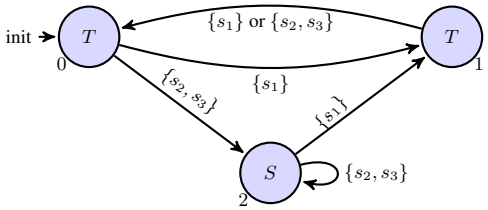

Fig. 2: Winning strategy synthesized by Acacia-K for Example 1

Example 2 (The 3-Coin Game). Another example that we tried is a game played using three coins which are arranged on a table with either *head* or *tail* up. The system doesn't see the coins, but knows at each time the number of tails and heads. Then, the game is infinitely played as follows. At the beginning the environment chooses an initial configuration and then at each round, the system chooses a coin and the environment has to flip that coin and inform the system about the new number of heads and tails. The objective of the system is to reach, at least once, the state in which all the coins have the *heads* up and to avoid all the time the state in which all the coins are *tails*. Depending on the initial number of tails up, the system may or may not have a winning strategy.

In order to model this, we considered an environment model whose states are labelled with atomic propositions c_1, c_2, c_3 for the three coins, which are not visible for the system, and two other variables b_1, b_0 which are visible and represent the bits encoding the number of *heads* in the configuration. The actions of the system are C_1, C_2, C_3 with which he chooses a coin and the environment has to flip the coin chosen by the system by playing only the action $done$. A picture of the environment is in [6].

Then, the specification is translated into the KLTL$^+$formula $K\Diamond(c_1 \wedge c_2 \wedge c_3) \wedge \Box K(c_1 \vee c_2 \vee c_3)$. Then, assuming that the initial state of the environment has two *heads*, the synthesized strategy proposes to "check" the position of every coin by double

flipping. If after one flip, the winning state is not reached, the system flips back the coin and at the third round he chooses another coin to check. A picture of the strategy can be found in [6]. For this example, Acacia-K constructs a UCT with 79 states, synthesises a strategy with 10 states, and the total running time is 3.9s.

Example 3 (n-Prisoners Enigma). Finally, the last example is about n prisoners in a prison, each one in his own cell and they cannot communicate. There is a special room with a light bulb and a switch and a policeman that, at each moment of time, sends only one prisoner in that room and gives him the possibility to turn on or off the light. The prisoners can only observe the light when they are in the special room. The guardians ensure that each prisoner is sent into the room an infinite number of times (fairness assumption). Before the game starts, the prisoners are allowed to communicate, and they know the initial state of the light. The goal of the prisoners is to learn whether all of them have visited the special room at least once – more specifically, whenever all prisoners have visited the room, one specially designated prisoner must know that fact.

Assume that the light is initially off. Then the winning strategy is that the special prisoner, say prisoner n, will count up to $n-1$. For all $1 \leq j \leq n-1$, the fairness assumption ensures that prisoner j will visit the room again and again until the game stops. The first time j visits the room and the light is off, he turns it on, otherwise he does nothing. Prisoner n will turn the light off next time he enters the room, and increment his counter by 1. When the counter reaches $n-1$, prisoner n will be sure that all prisoners have visited the room at least once.

We have tried 3/4/5/6 prisoners versions (including the protagonist) of this problem, obtaining a one hour timeout for 6 agents. The statistics we obtained are the following:

Pris #	$\lvert\mathcal{M}_E\rvert$	$\lvert UCT\rvert$	$\lvert tb-UCT\rvert$	Aut constr (s)	$\lvert\mathcal{M}_\lambda\rvert$	Total time(s)
3	21	144	692	1.79s	12	1.87s
4	53	447	2203	1.98s	16	13.20s
5	129	1310	6514	199.06s	20	553.45s ($\simeq$ 9 min)
6	305	3633	18125	6081.69s	N/A	N/A

Again, Acacia-K generates strategies that are natural, the same that one would synthesize intuitively. For more details about this example see [6]. This fact is remarkable itself since, in synthesis, it is often a difficult task to generate small and natural strategies.

8 Conclusion

In this paper, we have defined a Safraless procedure for the synthesis of KLTL$^+$specifications in environment with imperfect information. This problem is 2ExpTime-c but we have shown that our procedure, based on universal co-Büchi tree automata, can be implemented efficiently thanks to an antichain symbolic approach. We have implemented a prototype and run some preliminary experiments that prove the feasibility of our method. While the UCT constructed by the tool are not small (around 1300 states), our tool can handle them, although in theory, the safety games could be exponentially larger than the UCT. Moreover, our tool synthesises small strategies that correspond to the intuitive strategies we would expect, although it goes through a non-trivial automata construction. As a future work, we want to see if Acacia-K scales well on larger examples. We also want to extend the tool to handle the full KLTL logic in an efficient way. This paper is an encouraging (and necessary) step towards this objective. In a first attempt to generalize the specifications, we plan to consider assume-guarantees specifications $K\phi \rightarrow \psi$, where ϕ is an LTL formula and ψ a KLTL$^+$ formula.

References

1. Acacia-k. Available at http://lacl.fr/~rbozianu/Acacia-K/.
2. D. Berwanger and L. Doyen. On the power of imperfect information. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS*, volume 2 of *LIPIcs*, pages 73–82. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.
3. R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Knighofer, M. Roveri, V. Schuppan, and R. Seeber. Ratsy a new requirements analysis tool with synthesis. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 425–429. Springer Berlin Heidelberg, 2010.
4. R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.
5. A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J.-F. Raskin. Acacia+, a tool for LTL synthesis. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 652–657. Springer, 2012.
6. R. Bozianu, C. Dima, and E. Filiot. Safraless synthesis for epistemic temporal specifications. Available at: `http://arxiv.org/abs/1405.0424`, 2014.
7. K. Chatterjee, L. Doyen, E. Filiot, and J.-F. Raskin. Doomsday equilibria for omega-regular games. In K. L. McMillan and X. Rival, editors, *VMCAI*, volume 8318 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2014.
8. K. Chatterjee, L. Doyen, and T. A. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, 43(2):268–284, 2013.
9. E. Filiot, N. Jin, and J.-F. Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.
10. B. D. Giampaolo, G. Geeraerts, J.-F. Raskin, and N. Sznajder. Safraless procedures for timed specifications. In *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010.
11. J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In T. Kameda, J. Misra, J. G. Peters, and N. Santoro, editors, *PODC*, pages 50–61. ACM, 1984.
12. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 117–124. IEEE Computer Society, 2006.
13. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE Computer Society, 2005.
14. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.
15. 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.
16. S. Schewe and B. Finkbeiner. Bounded synthesis. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 474–488. Springer, 2007.
17. R. van der Meyden and M. Vardi. Synthesis from knowledge-based specifications. In D. Sangiorgi and R. Simone, editors, *CONCUR'98 Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 1998.