Compositional Algorithms for LTL Synthesis

Emmanuel Filiot, Naiyong Jin, Jean-François Raskin

Compositional Algorithms for LTL Synthesis*

Emmanuel Filiot, Nayiong Jin, and Jean-François Raskin

CS, Université Libre de Bruxelles, Belgium

Abstract. In this paper, we provide two compositional algorithms to solve safety games and apply them to provide compositional algorithms for the LTL synthesis problem. We have implemented those new compositional algorithms, and we demonstrate that they are able to handle full LTL specifications that are orders of magnitude larger than the specifications that can be treated by the current state of the art algorithms.

1 Introduction

Context and motivations The *realizability problem* is best seen as a game between two players [14]. Given an LTL formula ϕ and a partition of its atomic propositions P into I and O, Player 1 starts by giving a subset $o_0 \subseteq O$ of propositions [1], Player 2 responds by giving a subset of propositions $i_0 \subseteq I$, then Player 1 gives o_1 and Player 2 responds by i_1, and so on. This game lasts forever and the outcome of the game is the infinite word $w = (i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2) \cdots \in (2^P)^\omega$. Player 1 wins if the resulting infinite word w is a model of ϕ. The *synthesis problem* asks to produce a winning strategy for Player 1 when the LTL formula is realizable. The LTL realizability problem is central when reasoning about specifications for reactive systems and has been studied starting from the end of the eighties with the seminal works by Pnueli and Rosner [14], and Abadi, Lamport and Wolper [1]. It has been shown 2EXPTIME-C in [15].[2] Despite their high worst-case computation complexity, we believe that it is possible to solve LTL realizability and synthesis problems in practice. We proceed here along recent research efforts that have brought new algorithmic ideas to attack this important problem.

Contributions In this paper, we propose two compositional algorithms to solve the LTL realizability and synthesis problems. Those algorithms rely on previous works where the LTL realizability problem for an LTL formula Φ is reduced to the resolution of a

* Work supported by the projects: (i) QUASIMODO (FP7- ICT-STREP-214755), Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded", `http://www.quasimodo.aau.dk/`, (ii) GASICS (ESF-EUROCORES LogiCCC), Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", `http://www.ulb.ac.be/di/gasics/`, (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", `http://moves.ulb.ac.be`, a PAI program funded by the Federal Belgian Gouvernment, and (iv) ECSPER (ANR-JC09-472677) and SFINCS (ANR-07-SESU-012), two projects supported by the French National Research Agency.

[1] Technically, we could have started with Player 2, for modelling reason it is conservative to start with Player 1.

[2] Older pioneering works consider the realizability problem but for more expressive and computationally intractable formalisms like MSO, see [19] for pointers.

safety game $G(\Phi)$ [7] (a similar reduction was proposed independently in [17] and applied to synthesis of distributed controllers). We show here that if the LTL specification has the form $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ i.e., a conjunction of LTL sub-specifications, then $G(\Phi)$ can be constructed and solved compositionally. The compositional algorithms are able to handle formulas that are several pages long while previous non-compositional algorithms were limited to toy examples.

The new algorithms rely on the following nice property of safety games: for any safety game G, there exists a function that maps each position of Player 1 to the set of all actions that are safe to play. We call this function the *master plan* of Player 1 in G. It encompasses all the winning strategies of Player 1. If Λ is the master plan of G then we denote by $G[\Lambda]$ the game G where the behavior of Player 1 is restricted by Λ.

To compute the winning positions of a safety game $G^{12} = G^1 \otimes G^2$ defined as the composition of two sub-games, we compute the master plans for the local components G^1 and G^2 before composition. Let Λ_1 (resp. Λ_2) be the master plan for G^1 (resp. G^2), then the winning positions in G^{12} are the same as the winning positions in $G^1[\Lambda_1] \otimes G^2[\Lambda_2]$. We develop a backward and a forward algorithms that exploit this property.

We have implemented the two compositional algorithms into our prototype Acacia and we provide an empirical evaluation of their performances on classical benchmarks and on a realistic case study taken from the IBM RuleBase tutorial[9]. This implementation is rather used to test the new concepts and to see how they behave for scalability test cases than to provide an advanced and deeply optimized prototype. In particular, our implementation is in Perl (as Lily [10]) and does not use BDDs.

Related works The first solution [14] to the LTL realizability and synthesis problem was based on Safra's procedure for the determinization of Büchi automata [16].

Following [12], the method proposed in our paper can be coined "Safraless" approach to the realizability and synthesis of LTL as it avoids the determinization (based on the Safra's procedure) of the automaton obtained from the LTL formula. Our approach, as the one proposed in [7], relies on a reduction to safety games.

In [12], Kupferman and Vardi proposed the first Safraless approach that reduces the LTL realizability problem to Büchi games, which has been implemented in the tool Lily [10]. In [13], a compositional approach to LTL realizability and synthesis is proposed. Their algorithm is based on a Safraless approach that transforms the synthesis problem into a Büchi and not a safety game as in our case. There is no notion like the master plan for Büchi games. To the best of our knowledge, their algorithm has not been implemented.

In [3], an algorithm for the realizability problem for a fragment of LTL, known as GR(1), is presented and evaluated on the case study of [9]. The specification into the GR(1) fragment for this case study is not trivial to obtain and so the gain in term of complexity[3] comes with a cost in term of expressing the problem in the fragment. Our approach is different as we want to consider the full LTL logic. In our opinion, it is important to target full LTL as it often allows for writing more declarative and more natural specifications.

In [18], the authors also consider LTL formulas of the form $\Phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$. They propose an algorithm to construct compositionally a parity game from such LTL specifications. Their algorithm uses a variant of Safra's determinization procedure and additionally tries to detect local parity games that are equivalent to safety games

[3] GR(1) has a better worst-case complexity than full LTL.

(because the associated LTL subformula is a safety property). For efficiently solving the entire game, they use BDDs.

In [11], a compositional algorithm is proposed for reasoning about network of components to control under partial observability. The class of properties that they consider is safety properties and not LTL properties. They propose a backward algorithm and no forward algorithm.

The implementation supporting the approaches described in [18] and [3] uses BDDs while our tool Acacia does not. While our algorithms could have been implemented with BDDs, we deliberately decided not to use them for two reasons. First, to fairly compare our Safraless approach with the one proposed in [12] and implemented in Lily, we needed to exclude BDDs as Lily does not use them. Second, several recent works on the efficient implementation of decision problems for automata shown that antichain based algorithms may outperform by several order of magnitude BDD implementations, see [5, 6] for more details.

2 Safety Games

In this section, we provide a definition of safety games that is well-suited to support our compositional methods detailed in the following sections. Player 1 will play the role of the system while Player 2 will play the role of the environment. This is why, as the reader will see, our definition of games is asymmetric.

Turn-based games A *turn-based game* on a finite set of moves $\mathsf{Moves} = \mathsf{Moves}_1 \uplus \mathsf{Moves}_2$ such that $\mathsf{Moves}_2 \neq \varnothing$ is a tuple $G = (S_1, S_2, \Gamma_1, \Delta_1, \Delta_2)$ where: (i) S_1 is the set of Player 1 positions, S_2 is the set of Player 2 positions, $S_1 \cap S_2 = \varnothing$, we let $S = S_1 \uplus S_2$. (ii) $\Gamma_1 : S_1 \to 2^{\mathsf{Moves}_1}$ is a function that assigns to each position of Player 1 the subset of moves that are available in that position. For Player 2, we assume that all the moves in Moves_2 are available in all the positions $s \in S_2$. (iii) $\Delta_1 : S_1 \times \mathsf{Moves}_1 \to S_2$ is a partial function, defined on pairs (s, m) when Player 1 chooses $m \in \Gamma_1(s)$, that maps (s, m) to the position reached from s. $\Delta_2 : S_2 \times \mathsf{Moves}_2 \to S_1$ is a function that maps (s, m) to the state reached from s when Player 2 chooses m.

We define the partial function Δ as the union of the partial function Δ_1 and the function Δ_2. Unless stated otherwise, we fix for the sequel of this section a turn-based game $G = (S_1, S_2, \Gamma_1, \Delta_1, \Delta_2)$ on moves $\mathsf{Moves} = \mathsf{Moves}_1 \uplus \mathsf{Moves}_2$.

Given a function $\Lambda : S_1 \to 2^{\mathsf{Moves}_1}$, the *restriction of G by Λ* is the game $G[\Lambda] = (S_1, S_2, \widehat{\Gamma_1}, \widehat{\Delta_1}, \Delta_2)$ where for all $s \in S_1$, $\widehat{\Gamma_1}(s) = \Gamma_1(s) \cap \Lambda(s)$ and $\widehat{\Delta_1}$ equals Δ_1 on the domain restricted to the pairs $\{(s, m) \mid s \in S_1 \wedge m \in \widehat{\Gamma_1}(s)\}$ i.e., $G[\Lambda]$ is as G but with the moves of Player 1 restricted by Λ.

Rules of the game The game on G is played in rounds and generates a finite or an infinite sequence of positions that we call a *play*. In the initial round, the game is in some position, say s_0, and we assume that Player 1 owns that position. Then if $\Gamma_1(s_0)$ is non-empty Player 1 chooses a move $m_0 \in \Gamma_1(s_0)$, and the game evolves to state $s_1 = \Delta_1(s_0, m_0)$, otherwise the game stops. If the game does not stop then the next round starts in s_1. Player 2 chooses a move $m_1 \in \mathsf{Moves}_2$ and the game proceeds to position $s_2 = \Delta_2(s_1, m_1)$. The game proceeds accordingly either for an infinite number of rounds or it stops when a position $s \in S_1$ is reached such that $\Gamma_1(s) = \varnothing$. Player 1 wins if the game does not stop otherwise Player 2 wins (safety winning condition). Our

variant of safety games are thus zero-sum games as usual. In particular, the positions $s \in S_1$ such that $\Gamma_1(s) \neq \varnothing$ are the safe positions of Player 1.

Plays and strategies We now define formally the notions of play, strategy, outcome of a strategy and winning strategies. Given a sequence $\rho = s_0 s_1 \dots s_n \dots \in S^* \cup S^\omega$, we denote by $|\rho|$ its length (which is equal to ω if ρ is infinite). We denote by $\mathsf{first}(\rho)$ the first element of ρ, and if ρ is finite, we denote by $\mathsf{last}(\rho)$ its last element.

A *play* in G is a finite or infinite sequence of positions $\rho = s_0 s_1 \dots s_n \dots \in S^* \cup S^\omega$ such that : (i) if ρ is finite then $\mathsf{last}(\rho) \in S_1$ and $\Gamma_1(\mathsf{last}(\rho)) = \varnothing$; (ii) ρ is consistent with the moves and transitions of G i.e., for all i, $0 \leq i \leq |\rho|$, we have that $s_{i+1} = \Delta(s_i, m)$ for some $m \in \Gamma_1(s_i)$ if $s \in S_1$, or $m \in \mathsf{Moves}_2$ if $s \in S_2$.We denote by $\mathsf{Plays}(G)$ the set of plays in G.

Given a set of finite or infinite sequences $L \subseteq S^* \cup S^\omega$, we write $\mathsf{Pref}_j(L)$, $j \in \{1, 2\}$, for the set of prefixes of sequences in L that end up in a position of Player j. Let $\perp$ be such that $\perp \notin \mathsf{Moves}$. A *strategy for Player 1* in G is a function $\lambda_1 : \mathsf{Pref}_1(\mathsf{Plays}(G)) \to \mathsf{Moves}_1 \cup \{\perp\}$ which is consistent with the set of available moves i.e., for all $\rho \in \mathsf{Pref}_i(\mathsf{Plays}(G))$, we have that: (i) $\lambda_1(\rho) \in \Gamma_1(\mathsf{last}(\rho)) \cup \{\perp\}$, and (ii) $\lambda_1(\rho) = \perp$ only if $\Gamma_1(\mathsf{last}(\rho)) = \varnothing$. A *strategy for Player 2* in G is a function $\lambda_2 : \mathsf{Pref}_2(\mathsf{Plays}(G)) \to \mathsf{Moves}_2$. Note that the codomain of a Player 2's strategy never contains $\perp$ as all the moves of Player 2 are allowed at any position, whereas the moves of Player 1 are restricted by Γ_1.

A play $\rho = s_0 s_1 \dots s_n \dots \in \mathsf{Plays}(G)$ is *compatible* with a strategy λ_j of Player j ($j \in \{1, 2\}$), if for all i, $0 \leq i < |\rho|$, if $s_i \in S_j$ then $s_{i+1} = \Delta_j(s_i, \lambda_j(s_0 s_1 \dots s_i))$.We denote by $\mathsf{outcome}(G, s, \lambda_j)$ the subset of plays in $\mathsf{Plays}(G)$ that are compatible with the strategy λ_j of Player j, and that start in s. We denote by $\mathsf{outcome}(G, \lambda_j)$ the set $\bigcup_{s \in S} \mathsf{outcome}(G, s, \lambda_j)$, and by $\mathsf{outcome}(G, s, \lambda_1, \lambda_2)$ the unique play that is compatible with both λ_1 and λ_2, and starts in s.

The *winning* plays for Player 1 are those that are infinite i.e., $\mathsf{Win}_1(G) = \mathsf{Plays}(G) \cap S^\omega$, or equivalently those that never reach an unsafe position $s \in S_1$ of Player 1 where $\Gamma_1(s) = \varnothing$. A strategy λ_1 is *winning* in G from s_{ini} iff $\mathsf{outcome}(G, s_{\mathsf{ini}}, \lambda_1) \subseteq \mathsf{Win}_1(G)$. A game with such a winning condition in mind is called *safety game*. We denote by $\mathsf{WinPos}_1(G)$ the subset of positions $s \in S$ in G for which there exists λ_1 such that $\mathsf{outcome}(G, s, \lambda_1) \subseteq \mathsf{Win}_1(G)$.

Games with initial position A *safety game with initial position* is a pair (G, s_{ini}) where $s_{\mathsf{ini}} \in S_1 \cup S_2$ is a position of the game structure G called the *initial position*. The set of plays in (G, s_{ini}) are the plays of G starting in s_{ini}, i.e. $\mathsf{Plays}(G, s_{\mathsf{ini}}) = \mathsf{Plays}(G) \cap s_{\mathsf{ini}} \cdot (S^* \cup S^\omega)$. All the previous notions carry over to games with initial positions.

Solving safety games The classical fixpoint algorithm to solve safety games relies on iterating the following monotone operator over sets of game positions. Let $X \subseteq S_1 \uplus S_2$:

$$\mathsf{CPre}_1(X) = \{s \in S_1 \mid \exists m \in \Gamma_1(s), \Delta_1(s, m) \in X\} \cup \{s \in S_2 \mid \forall m \in \mathsf{Moves}_2, \Delta_2(s, m) \in X\}$$

i.e., $\mathsf{CPre}_1(X)$ contains all the positions $s \in S_1$ from which Player 1 can force X in one step, and all the positions $s \in S_2$ where Player 2 cannot avoid X in one step. Now, we define the following sequence of subsets of positions:

$$W_0 = \{s \in S_1 \mid \Gamma_1(s) \neq \varnothing\} \cup S_2 \qquad W_i = W_{i-1} \cap \mathsf{CPre}(W_{i-1}) \text{ for all } i \geq 1$$

Denote by $W^\natural$ the fixpoint of this sequence. It is well known that $W^\natural = \mathsf{WinPos}_1(G)$.

Master plan Let $\Lambda_1 : S_1 \to 2^{\mathsf{Moves}_1}$ be defined as follows: for all $s \in S_1$, $\Lambda_1(s) = \{m \in \Gamma_1(s) \mid \Delta_1(s,m) \in W^\natural\}$ i.e., $\Lambda_1(s)$ contains all the moves that Player 1 can play in s in order to win the safety game. We call Λ_1 the *master plan* of Player 1 and we write it $\mathsf{MP}(G)$. The following lemma states that $\mathsf{MP}(G)$ can be interpreted as a compact representation of all the winning strategies of Player 1 in the game G:

Lemma 1. *For all strategies λ_1 of Player 1 in G, for all $s \in S$, λ_1 is winning in G from s iff λ_1 is a strategy in $(G[\mathsf{MP}(G)], s)$ and $\lambda_1(s) \neq \bot$.*

Now that we have defined and characterized the notion of master plan, we show that we can compute directly the master plan associated with a game using a variant of the CPre operator and sequence W. The variant of CPre considers the effect of some Player 1's move followed by some Player 2's move. Let $\widehat{\mathsf{CPre}} : (S_1 \to 2^{\mathsf{Moves}_1}) \to (S_1 \to 2^{\mathsf{Moves}_1})$ be defined as follows. For all $s \in S_1$, let:

$$\widehat{\mathsf{CPre}}(\Lambda)(s) = \{m \in \Lambda(s) \mid \forall m' \in \mathsf{Moves}_2 : \Lambda(\Delta_2(\Delta_1(s,m),m')) \neq \varnothing\}$$

Consider the following sequence of functions: $\Lambda_0 = \Gamma_1$, and $\Lambda_i = \widehat{\mathsf{CPre}}(\Lambda_{i-1})$, $i \geq 1$. This sequence stabilizes after at most $O(|S|)$ iterations and we denote by $\Lambda^\natural$ the function on which the sequence stabilizes. Clearly, the value on which the sequence stabilizes corresponds exactly to the master plan of G:

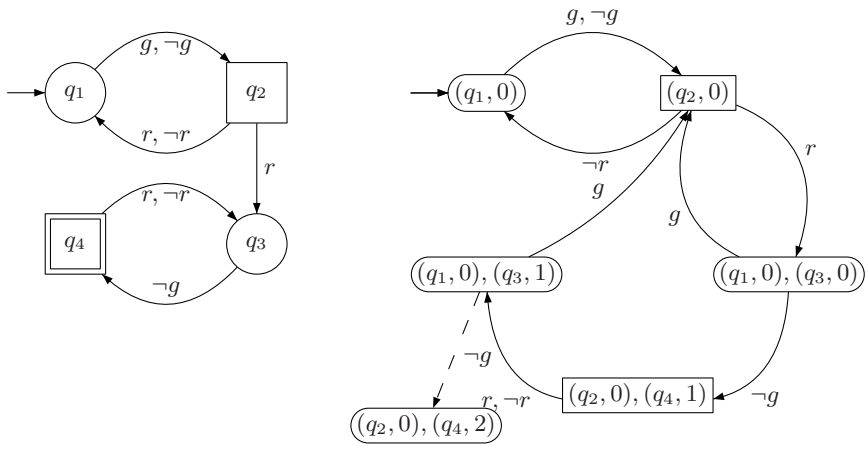
Theorem 1. $\Lambda^\natural = \mathsf{MP}(G)$.

3 From LTL realizability to safety games

In this section, after recalling the formal definition of the LTL realizability problem, we recall the essential results of [17, 7] where it is shown how to reduce the LTL realizability problem to a safety game problem.

Linear Temporal Logic (LTL) The formulas of LTL are defined over a set of atomic propositions P. The syntax is given by: $\phi ::= p \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{X}\phi \mid \phi \mathcal{U} \phi$ with $p \in P$. The notations true, false, $\phi_1 \wedge \phi_2$, $\Diamond\phi$ and $\Box\phi$ are defined as usual. LTL formulas ϕ are interpreted on infinite words $w = \sigma_0\sigma_1\sigma_2 \ldots \in (2^P)^\omega$ 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 Let P be a finite set of propositions. Unless otherwise stated, we partition P into I the set of *input signals* controlled by Player 2 (the environment), and O the set of *output signals* controlled by Player 1 (the controller). We let $\Sigma = 2^P$, $\Sigma_I = 2^I$, and $\Sigma_O = 2^O$. The realizability problem is best seen as a game. The players play according to strategies. A strategy for Player 1 is a (total) mapping $\lambda_1 : (\Sigma_O\Sigma_I)^* \to \Sigma_O$ while a strategy for Player 2 is a (total) mapping $\lambda_2 : \Sigma_O(\Sigma_I\Sigma_O)^* \to \Sigma_I$. The outcome of λ_1 and λ_2 is the word $\mathsf{outcome}(\lambda_1, \lambda_2) = (o_0 \cup i_0)(o_1 \cup i_1) \ldots$ such that for all $j \geq 0$, $o_j = \lambda_1(o_0 i_0 \ldots o_{j-1} i_{j-1})$ and $i_j = \lambda_2(o_0 i_0 \ldots o_{j-1} i_{j-1} o_j)$. In particular, $o_0 = \lambda_1(\epsilon)$ and $i_0 = \lambda_2(o_0)$. Given an LTL formula ϕ, the *realizability problem* is to decide whether there exists a strategy λ_1 of Player 1 such that for all strategies λ_2 of Player 2, $\mathsf{outcome}(\lambda_1, \lambda_2) \models \phi$. 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 [14]. The *synthesis problem* is to find a finite-state strategy that realizes the LTL specification.



(a) UCW (b) Safety game $G(\phi, 1)$

Fig. 1. UCW and safety game for the formula $\phi \equiv \Box(r \rightarrow \mathcal{X}\Diamond g)$

Universal CoBüchi automata LTL formulas are associated with *turn-based automaton* A over Σ_I and Σ_O. A turn-based automaton is a tuple $A = (\Sigma_I, \Sigma_O, Q_I, Q_O, Q_{\mathsf{ini}}, \alpha, \delta_I, \delta_O)$ where Q_I, Q_O are finite sets of input and output states respectively, $Q_{\mathsf{ini}} \subseteq Q_O$ is the set of initial states, $\alpha \subseteq Q_I \cup Q_O$ 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. Wlog we assume that the automata are *complete*, i.e. for all $t \in \{I, O\}$, all $q \in Q_t$ and all $\sigma \in \Sigma_t$, $\delta_t(q, \sigma) \neq \varnothing$. Turn-based automata A run on words $w = (o_0 \cup i_0)(o_1 \cup i_1) \cdots \in \Sigma^\omega$ as follows: a *run* on w is a word $\rho = \rho_0 \rho_1 \cdots \in (Q_O Q_I)^\omega$ such that $\rho_0 \in Q_{\mathsf{ini}}$ and for all $j \geq 0$, $(\rho_{2j}, o_j, \rho_{2j+1}) \in \delta_O$ and $(\rho_{2j+1}, i_j, \rho_{2j+2}) \in \delta_I$. Let $K \in \mathbb{N}$. We consider the universal co-Büchi (resp. K-co-Büchi) accepting condition, for which a word w is accepted iff any run on w visits finitely many (resp. at most K) accepting states. With the K-co-Büchi interpretation in mind, we say that (A, K) is a universal K-co-Büchi turn-based automaton. We denote by $L_{\text{uc}}(A)$ and $L_{\text{uc},K}(A)$ the languages accepted by A with these two accepting conditions resp. Turn-based automata with universal co-Büchi and K-co-Büchi acceptance conditions are denoted by UCW and UKCW. As they define set of infinite words, they can be taken as input to the realizability problem.

It is known that for any LTL formula one can construct an equivalent UCW A_ϕ (possibly exponentially larger) [20]. Fig. 1(a) represents a UCW equivalent to the formula $\Box(r \rightarrow \mathcal{X}(\Diamond g))$, where r is an input signal and g is an output signal. States of Q_O are denoted by circles while states of Q_I are denoted by squares. The transitions on missing letters are going to an additional sink non-accepting state that we do not represent for the sake of readability. If a request r is never granted, then a run will visit the accepting state q_4 infinitely often.

The realizability problem can be reduced from a UCW to a UKCW specification:

Theorem 2 ([17, 7]). *Let A be a* UCW *over Σ_I, Σ_O with n states and $K = 2n(n^{2n+2}+1)$. Then A is realizable iff (A, K) is realizable.*

Let us recall the intuition behind the correctness of this result. First, if the specification (A, K) is realizable then clearly the specification A is also realizable as $L_{\text{uc},K}(A) \subseteq$

$L_{\text{uc}}(A)$. Second, if the specification A is realizable then we know that there exists a finite memory strategy λ_1 that realizes it [14]. Any run on any outcome of λ_1 visits accepting states only a number of time equal to K, which is bounded by the size of the strategy. So λ_1 not only realizes the specification A but a stronger specification (A, K).

Reduction to safety game Clearly UKCW specifications are safety properties. The reduction to a safety game relies on the fact that UKCW can easily be made deterministic. Given a UKCW A, the game $G(A, K)$ is constructed via a subset construction extended with counters for each state 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 set of game positions S_1 for Player 1 is therefore the set of functions $F : Q_O$ to $\{-1, \ldots, K + 1\}$. The set S_2 is similarly defined as the functions $F : Q_I$ to $\{-1, \ldots, K + 1\}$. The set of moves of both players are the letters they can choose, i.e. $\mathsf{Moves}_1 = \Sigma_O$ and $\mathsf{Moves}_2 = \Sigma_I$. The set of available moves in a position are defined via a successor function succ such that for all $F \in S_i$ and $\sigma \in \mathsf{Moves}_i$,

$$\mathsf{succ}(F, \sigma) = q \mapsto \max\{\min(K + 1, F(p) + (q \in \alpha)) \mid q \in \delta(p, \sigma), F(p) \neq -1\}$$

where $\max \varnothing = -1$, and $(q \in \alpha) = 1$ if q is in α, and 0 otherwise. An action $\sigma_1 \in \mathsf{Moves}_1$ is available for Player 1 in a position $F \in S_1$ if the counters of F and $\mathsf{succ}(F, \sigma)$ do not exceed K. More formally, $\sigma \in \Gamma_1(F)$ iff for all $p \in Q_0$ and all $q \in Q_I$, $F(p) \leq K$ and $\mathsf{succ}(F, \sigma)(q) \leq K$. The transition function Δ_1 is defined by $\Delta_1(F, \sigma) = \mathsf{succ}(F, \sigma)$ for all $F \in S_1$ and all $\sigma \in \Gamma_1(s)$. The function Δ_2 is defined by $\Delta_2(F, \sigma) = \mathsf{succ}(F, \sigma)$ for all $F \in S_2$ and all $\sigma \in \mathsf{Moves}_2$. Finally, we start the game in the initial position $F_0 \in S_1$ such that for all $q \in Q_O$, $F(q) = -1$ if q is not initial, and 0 if q is initial but not final, and 1 if q is initial and final.

Associating a safety game with an LTL formula ϕ is done as follows: (1) construct a UCW A_ϕ equivalent to ϕ, (2) construct $G(A_\phi, K)$, denoted as $G(\psi, K)$ in the sequel, where $K = 2n(n^{2n+2} + 1)$ and n is the number of states of A_ϕ.

Incremental algorithm In practice, for checking the existence of a winning strategy for Player 1 in the safety game, we rely on an incremental approach. For all $K_1, K_2 \cdot 0 \leq K_1 \leq K_2$, if Player 1 can win $G(A, K_1)$, then she can win $G(A, K_2)$. This is because $L_{uc,K_1}(A) \subseteq L_{uc,K_2}(A) \subseteq L_{uc}(A)$. Therefore we can test the existence of strategies for increasing values of K. In all our examples (see Section 6), the smallest K for which Player 1 can win is very small (less than 5).

Example Fig. 1(b) represents the safety game (for $K = 1$) associated with the formula $\Box(r \rightarrow \mathcal{X}\Diamond g)$. Positions are pairs of states of the UCW with their counter values. Player 1's positions are denoted by circles while Player 2's positions are denoted by squares. The unavailable move of Player 1 from position $(q_2, 0)$ is denoted by a dashed arrow. It goes to a position where a counter exceeds the value K. The master plan of the game corresponds in this case to all the moves attached to plain arrows for Player 1's positions. Indeed Player 1 wins the game iff she never follows the dashed arrow.

Antichain-based symbolic algorithm In practice, we do not construct the game $G(A, K)$ explicitly, as it may be too large. However it has a nice structure that can be exploited for defining an efficient symbolic implementation for the computation of the sequence of W_i's defined in Section 2. The main idea is to consider an ordering on the positions in $G(A, K)$. 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 1 can win from

F' then she can also win from all $F \preceq F'$, since she has seen less accepting states in F than in F'. The consequence of this observation is that all the sets W_i are downward closed for the relation $\preceq$. It is shown in [7] that consequently all the computations can be done efficiently by manipulating only $\preceq$-maximal elements.

4 Compositional safety games

In this section, we define compositional safety games and develop two abstract compositional algorithms to solve such games.

Composition of safety games We now consider products of safety games. Let G^i, $i \in \{1, \ldots, n\}$, be n safety games $G^i = (S_1^i, S_2^i, \Gamma_1^i, \Delta_1^i, \Delta_2^i)$ defined on the same sets of moves $\mathsf{Moves} = \mathsf{Moves}_1 \uplus \mathsf{Moves}_2$. Their product, denoted by $\otimes_{i=1}^{i=n} G^i$, is the safety game $G^\otimes = (S_1^\otimes, S_2^\otimes, \Gamma_1^\otimes, \Delta_1^\otimes, \Delta_2^\otimes)$[4] defined as follows:

- $S_j^\otimes = S_j^1 \times S_j^2 \times \cdots \times S_j^n,\ j = 1, 2$;
- for $s = (s^1, s^2, \ldots, s^n) \in S_1^\otimes$, $\Gamma_1^\otimes(s) = \Gamma_1^1(s^1) \cap \Gamma_1^2(s^2) \cap \cdots \cap \Gamma_1^n(s^n)$;
- for $j \in \{1, 2\}$ and $s = (s^1, s^2, \ldots, s^n) \in S_j^\otimes$, let $m \in \Gamma_1^\otimes(s)$ if $j = 1$ or $m \in \mathsf{Moves}_2$ if $j = 2$. Then $\Delta_j^\otimes(s) = (t^1, t^2, \ldots, t^n)$, where $t^i = \Delta_j^i(s^i, m)$ for all $i \in \{1, 2, \ldots, n\}$;

Backward compositional reasoning We now define a backward compositional algorithm to solve the safety game $G^\otimes$. The correctness of this algorithm is justified by the following lemmas. For readability, we express the properties for composed games defined from two components. All the properties generalize to any number of components. The first part of the lemma states that to compute the master plan of a composition, we can first reduce each component to its *local* master plan. The second part of the lemma states that the master plan of a component is the master plan of the component where the choices of Player 1 has been restricted by one application of the $\widehat{\mathsf{CPre}}$ operator.

Lemma 2. (a) *Let* $G^{12} = G^1 \otimes G^2$, *let* $\Lambda_1 = \mathsf{MP}(G^1)$ *and* $\Lambda_2 = \mathsf{MP}(G^2)$ *then*

$$\mathsf{MP}(G^{12}) = \mathsf{MP}(G^1[\Lambda_1] \otimes G^2[\Lambda_2])$$

(b) *For any game* G, $\mathsf{MP}(G) = \mathsf{MP}(G[\widehat{\mathsf{CPre}}(\Gamma_1)])$.

Let $\Lambda : S_1^1 \times S_1^2 \times \cdots \times S_1^n \to 2^{\mathsf{Moves}}$, we let $\pi_i(\Lambda)$ the function with domain S_1^i and codomain 2^{Moves_1} such that for all $s \in S_1^i$, $\pi_i(\Lambda)(s)$ is the set of moves allowed by Λ in one tuple $(s^1, s^2, \ldots, s^n)$ such that $s^i = s$. Formally, $\pi_i(\Lambda)(s) = \bigcup\{\Lambda(s^1, s^2, \ldots, s^n) \mid (s^1, s^2, \ldots, s^n) \in S_1^\otimes,\ s^i = s\}$. Given two functions $\Lambda_1 : S_1 \to 2^{\mathsf{Moves}_1}$ and $\Lambda_2 : S_1 \to 2^{\mathsf{Moves}_1}$, we define $\Lambda_1 \cap \Lambda_2$ as the function on domain S_1 such that for all $s \in S_1$: $\Lambda_1 \cap \Lambda_2(s) = \Lambda_1(s) \cap \Lambda_2(s)$. Given two functions $\Lambda_1 : S_1 \to 2^{\mathsf{Moves}_1}$ and $\Lambda_2 : S_2 \to 2^{\mathsf{Moves}_1}$, we define $(\Lambda_1 \times \Lambda_2) : S_1 \times S_2 \to 2^{\mathsf{Moves}_1}$ as $(\Lambda_1 \times \Lambda_2)(s_1, s_2) = \Lambda_1(s_1) \cap \Lambda_2(s_2)$.

Based on Lemma 2, we propose the following compositional algorithm to compute the master plan of a safety game defined as the composition of local safety games. First, compute locally the master plans of the components. Then compose the local master plans and apply one time the $\widehat{\mathsf{CPre}}$ operator to this composition. This application of $\widehat{\mathsf{CPre}}$ compute a new function Λ that contains information about the one-step

[4] Clearly, the product operation is associative up to isomorphism.

inconsistencies between local master plans. Project back on the local components the information gained by the function Λ, and iterate. This is formalized in Algorithm 1 whose correctness is asserted by Theorem 3.

Algorithm 1: Backward composition

Data: $G^{\otimes} = G^1 \otimes G^2 \otimes \cdots \otimes G^n$
$\Lambda \leftarrow \Gamma_1^{\otimes}$;
repeat
  $\Lambda^i := \mathsf{MP}(G^i[\pi_i(\Lambda)]), 1 \leq i \leq n$;
  $\Lambda := \widehat{\mathsf{CPre}}(\Lambda \cap (\Lambda^1 \times \cdots \times \Lambda^n))$
until *Λ does not change*;
return Λ

Algorithm 2: Forward composition

Data: $G^{\otimes} = G^1 \otimes G^2 \otimes \cdots \otimes G^n$
$\Lambda^i := \mathsf{MP}_{\mathsf{Reach}}(G^i, s^i_{\mathsf{ini}}), 1 \leq i \leq n$;
$\Lambda := \mathsf{MP}_{\mathsf{Reach}}(G^1[\Lambda^1] \otimes \cdots \otimes G^n[\Lambda^n], (s^1_{\mathsf{ini}}, s^2_{\mathsf{ini}}, \ldots, s^n_{\mathsf{ini}}))$
return Λ

Theorem 3. *The value Λ returned by Algorithm 1 is equal to* $\mathsf{MP}(G^{\otimes})$.

Forward compositional reasoning When solving safety games, we may be interested only in computing winning strategies for a fixed starting position, say s_{ini}. In this case, the value of the master plan is not useful for positions that are not reachable when playing winning strategies from s_{ini}. So, we are interested in computing a master plan only for the winning and *reachable* positions. Given a game G and a state s_{ini}, we denote by $\mathsf{Reach}(G, s_{\mathsf{ini}})$ the subset of positions that are reachable from s_{ini} in G i.e., the states s' such that there exists a finite sequence $s_0 s_1 \ldots s_n$ with $s_0 = s_{\mathsf{ini}}$, $s_n = s'$ and for all i, $0 \leq i < n$, there exists $m \in \Gamma_1(s_i) \cup \mathsf{Moves}_2$ such that $s_{i+1} = \Delta(s_i, m)$. The *master plan of reachable positions* for (G, s_{ini}), denoted by $\mathsf{MP}_{\mathsf{Reach}}(G, s_{\mathsf{ini}})$ is defined for all $s \in S$ as follows:

$$\mathsf{MP}_{\mathsf{Reach}}(G, s_{\mathsf{ini}})(s) = \begin{cases} \mathsf{MP}(G)(s) & \text{if } s \in \mathsf{Reach}(G[\Lambda], s_{\mathsf{ini}}) \\ \varnothing & \text{otherwise.} \end{cases}$$

The following lemma shows that for a game defined compositionally, its master plan can also be defined compositionally. For readability we express the lemma only for two components but, as for the previous lemmas, it extends to any number of components:

Lemma 3. *Let $\Lambda_1 = \mathsf{MP}_{\mathsf{Reach}}(G^1, s^1_{\mathsf{ini}})$ and $\Lambda_2 = \mathsf{MP}_{\mathsf{Reach}}(G^2, s^2_{\mathsf{ini}})$.*

$$\mathsf{MP}_{\mathsf{Reach}}(G^1 \otimes G^2, (s^1_{\mathsf{ini}}, s^2_{\mathsf{ini}})) = \mathsf{MP}_{\mathsf{Reach}}(G^1[\Lambda_1] \otimes G^2[\Lambda_2], (s^1_{\mathsf{ini}}, s^2_{\mathsf{ini}}))$$

As composition of safety games is an associative operator, we can use variants of the algorithm above where we first compose some of the components and compute their master plan of reachable positions before doing the global composition.

To efficiently compute the master plan of reachable positions of a game G, we use the OTFUR algorithm of [4]. We say that a position $s \in S_1$ is *unsafe* if $\Gamma_1(s) = \varnothing$, or all its successors are unsafe. A position $s \in S_2$ is unsafe if one of its successors is unsafe. The algorithm explores the state space by starting from the initial state in a forward fashion. When sufficient information is known about the successors of a position s, it back-propagates the unsafe information to s. At the end of the algorithm, the master plan which allows all moves that lead to a safe position is exactly $\mathsf{MP}_{\mathsf{Reach}}(G, s_{\mathsf{ini}})$. Fig. 2 illustrates the result of the OTFUR algorithms applied on the product of two safety games G_1, G_2 over the possible moves o_1, o_2, o_3 for Player 1 and i_1, i_2 for Player 2. We assume that G_1, G_2 contains only winning actions, i.e. $G_i = G_i[\mathsf{MP}(G_i)]$ for all $i = 1, 2$. The master plan of reachable states for $G_1 \otimes G_2$ corresponds to plain arrows.

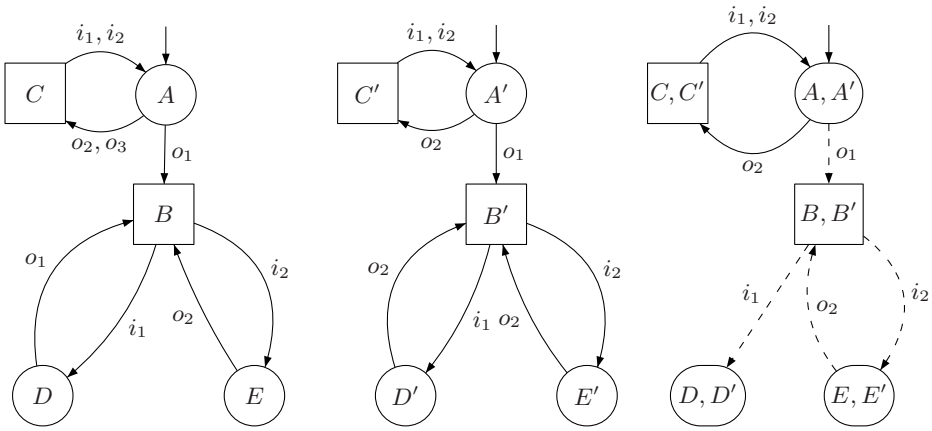


Fig. 2. Two games and their common master plan of reachable states

Dashed arrows are those which have been traversed during the OTFUR algorithm but have been removed due to backpropagation of unsafe information. From node $\langle A, A' \rangle$ the move o_3 is not a common move, therefore o_3 is not available in the product as well. However o_2 is available in both games and leads to C and C' respectively. Similarly, o_1 is available in both games and goes to $\langle B, B' \rangle$. From $\langle B, B' \rangle$ one can reach $\langle D, D' \rangle$ by i_1 but from $\langle D, D' \rangle$ there is no common action. Therefore $\langle D, D' \rangle$ is unsafe. Since one of the successor of $\langle B, B' \rangle$ is unsafe and $\langle B, B' \rangle$ is owned by Player 2, $\langle B, B' \rangle$ is declared to be unsafe as well. All the remaining moves are winning in the $G_1 \otimes G_2$, as they are winning both in G_1 and G_2.

Remark 1. It should be noted that each Λ^i in Alg. 2 can be replaced by the full *master plan* without changing the output of the forward algorithm. Indeed, it is easy to see that $\mathsf{Reach}(G[\mathsf{MP}_{\mathsf{Reach}}(G, s_{\mathsf{ini}})], s_{\mathsf{ini}}) = \mathsf{Reach}(G[\mathsf{MP}(G)], s_{\mathsf{ini}})$. So, we can mix the backward and forward algorithms. For instance, we can compute locally the master plan of each G^i using the backward algorithm of [7], and then check global realizability using the OTFUR algorithm.

5 Compositional LTL Synthesis and Dropping Assumptions

In this section, we show how to define compositionally the safety game associated with an LTL formula when this formula is given as a conjunction of subformulas i.e., $\psi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$. Assume from now on that we have fixed some $K \in \mathbb{N}$. We first construct for each subformula ϕ_i the corresponding UKCW A_{ϕ_i} on the alphabet of ψ[5], and their associated safety games $G(\phi_i, K)$. The game $G(\psi, K)$ for the conjunction ψ is isomorphic to the game $\otimes_{i=1}^{i=n} G(\phi_i, K)$.

To establish this result, we rely on a notion of product at the level of turn-based automata. Let $A_i = (\Sigma_I, \Sigma_O, Q_I^i, Q_O^i, q_0^i, \alpha^i, \delta_I^i, \delta_O^i)$ for $i \in \{1, 2\}$ be two turn-based automata, then their product $A_1 \otimes A_2$ is the turn-based automaton defined as $(\Sigma_I, \Sigma_O, Q_I^1 \uplus Q_I^2, Q_O^1 \uplus Q_O^2, Q_{\mathsf{ini}}^1 \uplus Q_{\mathsf{ini}}^2, \alpha_1 \uplus \alpha_2, \delta_I^1 \uplus \delta_I^2, \delta_O^1 \uplus \delta_O^2)$. As we use universal interpretation i.e., we require all runs to respect the accepting condition, it is clear

[5] It is necessary to keep the entire alphabet when considering the subformulas to ensure proper definition of the product of games that asks for components defined on the same set of moves.

that executing the $A_1 \otimes A_2$ on a word w is equivalent to execute both A_1 and A_2 on this word. So w is accepted by the product iff it is accepted by each of the automata.

Proposition 1. *Let A_1 and A_2 be two* UCW *on the alphabet $\Sigma_1 \uplus \Sigma_2$, and $K \in \mathbb{N}$: (i) $L_{uc}(A_1 \otimes A_2) = L_{uc}(A_1) \cap L_{uc}(A_2)$, (ii) $L_{uc,K}(A_1 \otimes A_2) = L_{uc,K}(A_1) \cap L_{uc,K}(A_2)$*

As the state space and transition relation of $A_1 \otimes A_2$ is the disjunct union of the space spaces and transition relations of A_1 and A_2, the determinization of $A_1 \otimes A_2$ for a fixed $K \in \mathbb{N}$ is equivalent to the synchronized product of the determinizations of A_1 and A_2 for that K, and so we get the following theorem.

Theorem 4. *Let $\psi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$, $K \in \mathbb{N}$, $G(\psi, K)$ is isomorphic to $\otimes_{i=1}^{i=n} G(\phi_i, K)$.*

Even if it is natural to write large LTL specifications as conjunctions of subformulas, it is also sometimes convenient to write specifications that are of the form $(\bigwedge_{i=1}^{i=n} \psi_i) \rightarrow (\bigwedge_{j=1}^{j=m} \phi_j)$ where ψ_i's formalize a set of assumptions made on the environment (Player 2) and ϕ_j's formalize a set of guarantees that the system (Player 1) must enforce. In this case, we rewrite the formula into the logical equivalent formula $\bigwedge_{j=1}^{j=m}((\bigwedge_{i=1}^{i=n} \psi_i) \rightarrow \phi_j)$ which is a conjunction of LTL formulas as needed for the compositional construction described above. As logical equivalence is maintained, realizability is maintained as well.

Unfortunately, this formula is larger than the original formula as all the n assumptions are duplicated for all the m guarantees. But, the subformulas $(\bigwedge_{i=1}^{i=n} \psi_i) \rightarrow \phi_j, j \in \{1, \ldots, m\}$ are usually such that to guarantee ϕ_j, Player 1 does not need all the assumptions on the left of the implication. It is thus tempting to remove those assumptions that are *locally* unnecessary in order to get smaller local formulas. In practice, we apply the following rule. Let $\psi_1 \wedge \psi_2 \rightarrow \phi$ be a local formula such that ψ_2 and ϕ do not share common propositions then we replace $\psi_1 \wedge \psi_2 \rightarrow \phi$ by $\psi_1 \rightarrow \phi$. This simplification is correct in the following sense: if the formula obtained after dropping some assumptions in local formulas is realizable then the original formula is also realizable. Further, a Player 1's strategy to win the game defined by the simplified formula is also a Player 1's strategy to win the game defined by the original formula. This is justified by the fact that the new formula logically implies the original formula i.e. $\psi_1 \rightarrow \phi$ logically implies $\psi_1 \wedge \psi_2 \rightarrow \phi$. However, this heuristic is not complete because the local master plans may be more restrictive than necessary as we locally forget about global assumptions that exist in the original formula. We illustrate this on two examples.

Let $I = \{\mathsf{req}\}$, $O = \{\mathsf{grant}\}$ and $\phi = (\Box\Diamond\mathsf{req}) \rightarrow \Box\Diamond\mathsf{grant}$. In this formula, the assumption $\Box\Diamond\mathsf{req}$ is not relevant to the guarantee $\Box\Diamond\mathsf{grant}$. Realizing ϕ is thus equivalent to realizing $\Box\Diamond\mathsf{grant}$. However, the set of strategies realizing ϕ is not preserved when dropping the assumption. Indeed, the strategy that outputs a grant after each req realizes ϕ but it does not realize $\Box\Diamond\mathsf{grant}$, as this strategy relies on the behavior of the environment. Thus dropping assumption is weaker than the notion of *open implication* of [8], which requires that the strategies realizing ϕ have to realize $\Box\Diamond\mathsf{grant}$.

As illustrated by the previous example, dropping assumption does not preserve the set of strategies that realize the formula. Therefore, it can be the case that a realizable formula cannot be shown realizable with our compositional algorithm after locally dropping assumptions. In addition, it can be the case that a formula becomes unrealizable after dropping local assumptions. Consider for instance the formula $\phi =$

$\Box\Diamond \mathsf{req} \rightarrow (\Box\Diamond \mathsf{grant} \wedge \Box(\mathcal{X}(\neg \mathsf{grant}) \; \mathcal{U} \; \mathsf{req}))$. This formula is realizable, for instance by the strategy which outputs a grant iff the environment signal at the previous tick was a req. Other strategies realize this formula, like those which grant a request every n req signal (n is fixed), but all the strategies that realize ϕ have to exploit the behavior of the environment. Thus there is no strategy realizing the conjunction of $\Box\Diamond$grant and ϕ. Consequently, when we decompose ϕ into $\Box\Diamond \mathsf{req} \rightarrow \Box\Diamond \mathsf{grant}$ and $\Box\Diamond \mathsf{req} \rightarrow \Box(\mathcal{X}(\neg \mathsf{grant}) \; \mathcal{U} \; \mathsf{req})$, we must keep $\Box\Diamond$req in the two formulas.

Nevertheless, in our experiments, the dropping assumption heuristic is very effective and except for one example, it always maintains *compositional realizability*.

Symbolic compositional synthesis with antichains As mentioned in Sec. 3, we do not construct the games explicitely, but solve them on-the-fly by compactly representing by antichains the set of positions manipulated during the fixpoint computation. In particular, suppose that we are given a conjunction of formulas $\phi_1 \wedge \phi_2$, and some $K \in \mathbb{N}$. For all $i \in \{1, 2\}$, we first solve the subgame $G(\phi_i, K)$ by using the backward fixpoint computation of [7] and get the downward closed set of winning positions (for Player 1), represented by antichains. Some winning positions are owned by Player 1 (resp. Player 2), let this set be $\downarrow W_1$ (resp. $\downarrow W_2$), the downward closure of an antichain W_1 (resp. W_2). Then W_1 and W_2 also provide a compact representation of $\mathsf{MP}(G(\phi_i, K))$. Indeed, let F be a Player 1's position in $G(\phi_i, K)$, then $\mathsf{MP}(G(\phi_i, K))(F)$ is empty if $F \notin \downarrow W_1$ (the downward closure of W_1), otherwise is the set of moves $\sigma \in \Sigma_O$ such that $\mathsf{succ}(F, \sigma) \in \downarrow W_2$. This symbolic representation is used in practice for the forward and backward compositional algorithms (Algorithms 1 and 2 of Sec. 4).

Moreover, the partial order on game positions can also be exploited by the OTFUR algorithm of Section 4 used in the forward compositional algorithm. Indeed let F be some Player 1's position of some game $G(\phi, k)$. Clearly, F is loosing (for Player 1 the controller) iff all its minimal successors are loosing. We get the dual of this property when F is a position owned by Player 2 (the environment). In this case F is loosing (for the controller) iff one of its maximal successors is loosing. Therefore to decide whether a position is loosing, depending on whether it is a controller or an environment position, we have to visit its minimal or its maximal successors only. In the OFTUR algorithm, this is done by adding to the waiting list only the edges (s', s'') such that s'' is a minimal (or maximal) successor of s'. In the case of a position owned by the controller, we can do even better. Indeed, we can add only one minimal successor in the waiting list at a time. If it turns out that this successor is loosing, we add another minimal successor. Among the minimal successors, the choice is done as follows: we prefer to add an edge (s', s'') such that s'' has already been visited. Indeed, this potentially avoids unnecessary developments of new parts of the game. Note however that this optimization cannot be used to compute the master plan of reachable positions, but only some winning strategy, as some parts of the game may not be explored. In the experiments, we use the backward algorithm to solve the local games and the optimized forward algorithm to solve the global game.

6 Experimental evaluation

The compositional algorithms have been implemented in our prototype ACACIA [7]. The performances are evaluated on the examples provided with the tool LILY and on a larger specification of a buffer controller inspirated by the IBM rulebase tutorial [9].

Lily's test cases and parametric example We compare several methods on the realizable examples provided with LILY and on the parametric example of [7]. In those benchmarks, the formulas are of the form $\bigwedge_{i=1}^{i=n} \psi_i \rightarrow \bigwedge_{j=1}^{j=m} \phi_j$ where $\bigwedge_{i=1}^{i=n} \psi_i$ are a set of *assumptions* and $\bigwedge_{j=1}^{j=m} \phi_j$ are a set of *guarantees*. We decompose such formula into several pieces $(\bigwedge_{i=1}^{i=n} \psi_i) \rightarrow \phi_j$, as described in the previous section.

We compare four synthesis methods (Table 1). The first is the monolithic backward method of [7]. The second is the monolithic forward method based on the OTFUR algorithm optimized with antichains. The third method is a compositional method where the local games are solved with the backward algorithm of [7] and the global game with the forward algorithm OTFUR (optimized with antichains). Finally, the last method is the third method where we use the dropping assumption heuristic. For each method, we give the size of the automata (in the case of compositional methods it is the sum of the sizes of every local automata), the time to construct them, the time to check for realizability (*Check Time*), and the total time. The values in bold face are the best total times among all methods.

On small examples, we can see that the benefit of the compositional approach is not big (and in some cases the monolithic approach is even better). However for bigger formulas (demo 3.2 to 3.7), decomposing the formulas decreases the time to construct the automata, and the total realizability time is therefore better.

Now, we evaluate the benefit of dropping assumptions (last group of columns). For those experiments, we only consider the subset of formulas for which this heuristic can be applied. Our dropping heuristic does not work for demo 9 as it becomes unrealizable after the application of dropping assumptions. As we see in the table, the benefit of dropping assumptions is important and is growing with the size of the formulas that are considered. The compositional algorithms outperform the monolithic ones when combined with dropping assumptions. They also show promises for better scalability. This is confirmed by our next benchmark.

A realistic case study Now, we consider a set of realistic formulas (Table 2). All those formulas are out of reach of the monolithic approach as even the Büchi automaton for the formula cannot be constructed with state of the art tools. The generalized buffer (GenBuf) originates from the IBM's tutorial for her RuleBase verification tool. The benchmark has also the nice property that it can be scaled up by increasing the number of receivers in the protocol. In this case study, the formulas are of the form $\bigwedge_{i=1}^{i=n} \psi_i \rightarrow \phi_j$ and so they are readily amenable to our compositional algorithms.

In this case study, formulas are large: for example, the sum of the number of states in the UCW of the components is 96 for $\mathsf{gb}(s_2, r_2)$, and 2399 states for $\mathsf{gb}(s_2, r_7)$. Note that the tool Wring cannot handle $\mathsf{gb}(s_2, r_2)$ monolithically.

This case study allows us to illustrate the effect of different strategies for exploiting associativity of the product operation. In particular, we use different ways of parenthesizing the local games. In all those examples, the local games and intermediate combination of local games are solved with the backward compositional algorithm, while the last compositional step (at the top) is done with the forward method. In each strategy we first compute the master plan of each sub-formula. Then the column Flat refers to the strategy that check global realizability directly. The column Binary refers to the strategy that computes global realizability incrementally using the binary tree of sub-formulas. Finally, the column Heuristic refers to the strategy that computes global realizability incrementally using a specific tree of sub-formula defined by the user. The

column UCW_OPT refers to the time to optimize the automata with Lily's optimizations (this time was included in the UCW time in Table 1).

	Monolothic						Compositional				Compositional + DA			
			BACKWARD (Acacia'09)		FORWARD		FORWARD(global) BACKWARD(local)				FORWARD(global) BACKWARD(local)			
examples	|tbUCW| (states)	tbUCW Time(s)	Check Time(s)	Total time(s)	Check Time(s)	Total time(s)	Σ_i |tbUCW$_i$|	tbUCW Time(s)	Check Time(s)	Total time(s)	Σ_i |tbUCW$_i$|	tbUCW Time(s)	Check Time(s)	Total time(s)
3	20	0.49	0.00	0.49	0.01	0.5	28	0.40	0.01	0.41	17	0.06	0.00	**0.06**
5	26	0.71	0.00	0.71	0.01	0.72	42	0.70	0.02	0.72	34	0.40	0.02	**0.42**
6	37	1.22	0.02	1.24	0.02	1.24	57	1.14	0.03	1.17	45	0.79	0.06	**0.85**
7	22	0.60	0.00	0.60	0.01	0.61	41	0.66	0.02	0.68	33	0.40	0.02	**0.42**
9	13	0.13	0.01	0.14	0.00	**0.13**	31	0.26	0.00	0.26	na	na	na	na
13	7	0.00	0.00	0.00	0.01	0.01	4	0.01	0.00	**0.01**	na	na	na	na
14	14	0.11	0.00	0.11	0.01	**0.12**	27	0.77	0.01	0.78	15	0.03	0.00	0.03
15	16	0.06	0.02	0.08	0.00	**0.06**	22	0.11	0.03	0.14	na	na	na	na
16	21	0.22	0.31	0.53	0.07	**0.29**	45	0.20	0.14	0.34	na	na	na	na
17	17	0.16	0.04	0.20	0.03	**0.19**	23	0.16	0.05	0.21	na	na	na	na
18	22	0.34	0.21	0.55	0.19	**0.53**	45	0.35	0.16	0.51	na	na	na	na
19	18	0.31	0.01	0.32	0.01	0.32	27	0.25	0.03	0.28	27	0.26	0.01	**0.27**
20	105	2.67	0.01	2.68	0.01	2.68	154	2.43	0.03	2.46	101	1.52	0.02	**1.54**
21	27	7.38	0.22	7.60	0.28	7.66	43	1.40	0.52	1.92	44	0.55	0.51	**1.06**
22	45	7.08	0.03	7.11	0.02	7.1	80	10.26	0.05	10.31	49	1.51	0.13	**1.64**
3.2	36	0.94	0.02	0.96	0.00	0.94	40	0.79	0.02	**0.81**	na	na	na	na
3.3	56	1.80	0.15	1.95	0.02	1.82	60	1.21	0.06	**1.27**	na	na	na	na
3.4	84	3.12	1.24	4.36	0.04	3.16	80	1.63	0.10	**1.73**	na	na	na	na
3.5	128	3.52	9.94	13.46	0.12	3.64	100	2.04	0.17	**2.21**	na	na	na	na
3.6	204	10.22	100	110.22	0.46	10.68	120	2.40	0.39	**2.79**	na	na	na	na
3.7	344	26.48	660	686.48	2.35	28.82	140	2.96	1.02	**3.98**	na	na	na	na

Table 1. Performance comparison on Lily's benchmark and parametric example

Conclusion We have provided compositional algorithms for full LTL synthesis. Our algorithm are able to handle formulas that are several pages long (see [2]). We believe that our compositional approach is an essential step to make realizability check more practical. As future works, we plan to improve our tool by considering symbolic data-structures. Currently, the alphabet of signals is handled enumeratively and we believe that substantial gain could be obtain by handling it symbolically. This algorithmic improvement is orthogonal to the ones presented in this paper. It should be noted that the compositional approach we propose is general and can be applied, for example, if some sub-games are not specified using LTL but constructed directly from another specification language. This is important as in practice some modules could be easily specified directly by deterministic automata instead of LTL. Exploring the use of such mixed specification methodology is part of our future works.

					FLAT	BINARY	HEURISTIC	
	k	Σ|tbUCW|	tbUCW Time(s)	UCW_OPT Time(s)	Check Time(s)	Check Time(s)	Check Time(s)	|Moore machine|
gb_s2_r2	2	91	4.83	0.08	0.84	0.99	0.98	54
gb_s2_r3	2	150	8.52	0.17	7.33	36.27	6.99	63
gb_s2_r4	2	265	15.64	0.53	36.88	125.60	24.19	86
gb_s2_r5	2	531	26.48	2.11	154.02	266.36	70.41	107
gb_s2_r6	2	1116	50.70	14.38	889.12	1164.44	335.44	132
gb_s2_r7	2	2399	92.01	148.46	2310.74	timeout	1650.83	149

Table 2. Performance comparison on a scalability test for the forward methods

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP*, LNCS 372:1–17, 1989.
2. Acacia. Available at `http://www.antichains.be/acacia`, 2009.
3. R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from psl. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.
4. F. Cassez, A. David, E. Fleury, K.G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, LNCS 3653:66–80, 2005.
5. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A new algorithm for checking universality of finite automata. In *CAV*, LNCS 4144:17–30, 2006.
6. Doyen, L., Raskin, J.F.: Improved algorithms for the automata-based approach to model-checking. In *TACAS*, LNCS 4424:451–465, 2007.
7. E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *CAV*, LNCS 5643:263–277, 2009.
8. K. Greimel, R. Bloem, B. Jobstmann, and M. Y. Vardi. Open implication. In *ICALP'08*, LNCS 5126:361–372, 2008.
9. `www.research.ibm.com/haifa/projects/verification/RB_Homepage/tutorial3/`
10. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, pp 117–124. IEEE.
11. W. Kuijper and J. van de Pol. Compositional control synthesis for partially observable systems. In *CONCUR*, LNCS 5710:431–447, 2009.
12. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *FOCS*, pp 531–542, 2005, IEEE.
13. O. Kupferman, N. Piterman, and M. Y. Vardi. Safraless compositional synthesis. In *CAV*, LNCS 4144:31–44, 2006.
14. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pp 179–190, 1989, ACM.
15. R. Rosner. *Modular synthesis of reactive systems*. Ph.d. dissertation, Weizmann Institute of Science, 1992.
16. S. Safra. On the complexity of ω automata. In *FOCS*, pp 319–327, 1988.
17. S. Schewe and B. Finkbeiner. Bounded synthesis. In *ATVA*, LNCS 4762:474–488. 2007.
18. S. Sohail and F. Somenzi. Safety first: A two-stage algorithm for ltl games. In *FMCAD*, pp 77–84, 2009, IEEE.
19. W. Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science*, LNCS 4800:635–655, 2008.
20. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, LNCS 1043:238–266, 1995.