Distributed Synthesis for LTL Fragments

Krishnendu Chatterjee, Thomas A. Henzinger, Jan Otop, Andreas Pavlogiannis
IST Austria
{chatterjee, tah, jotop, pavlogiannis}@ist.ac.at

Abstract—We consider the distributed synthesis problem for temporal logic specifications. Traditionally, the problem has been studied for LTL, and the previous results show that the problem is decidable iff there is no information fork in the architecture. We consider the problem for fragments of LTL and our main results are as follows: (1) We show that the problem is undecidable for architectures with information forks even for the fragment of LTL with temporal operators restricted to next and eventually. (2) For specifications restricted to globally along with non-nested next operators, we establish decidability (in EXPSPACE) for star architectures where the processes receive disjoint inputs, whereas we establish undecidability for architectures containing an information fork-meet structure. (3) Finally, we consider LTL without the next operator; and establish decidability (NEXPTIME-complete) for all architectures for a fragment that consists of a set of safety assumptions, and a set of guarantees where each guarantee is a safety, reachability, or liveness condition.

I. INTRODUCTION

Synthesis and distributed synthesis. The synthesis problem is the most rigorous form of systems design, where the goal is to construct a system from a given temporal logic specification. The problem was originally proposed by Church [1] for synthesis of circuits, and has been revisited in many different contexts, such as supervisory control of discrete event systems [2], synthesis of reactive systems [3], and several others. In a seminal work, Pnueli and Rosner [4] extended the classical synthesis problem to a distributed setting. In the distributed synthesis problem, the input consists of (i) an architecture of synchronously communicating processes, that exchange messages through communication channels; and (ii) a specification given as a temporal logic formula; and the synthesis question asks for a reactive system for each process such that the specification is satisfied. The most common logic to express the temporal logic specification is the linear-time temporal logic (LTL) [5].

Previous results for distributed synthesis for LTL. In general the distributed synthesis problem is undecidable for LTL, but the problem is decidable for pipeline architectures [4]. The undecidability proof uses ideas originating from the undecidability proof of three-player imperfect-information games [6], [7]. The decidability results for distributed synthesis were extended to other similar architectures, such as one-way rings [8], and also a distributed games framework was proposed in [9]. Finally, a complete topological criterion on the architecture for decidability of distributed synthesis for LTL was presented [10], where it was shown that the problem is decidable if and only if there is no information fork in the underlying architecture. Architectures without information forks can essentially be reduced to pipelines.

Fragments of LTL. While LTL provides a very rich framework to express temporal logic specifications, in recent years, several fragments of LTL have been considered for efficient synthesis of systems in the non-distributed setting. Such fragments often encompass a large class of properties that arise in practice and admit efficient synthesis algorithms, as compared to the whole LTL. In [11], [12] the authors considered a fragment of LTL with only eventually (reachability) and globally (safety) as the temporal operators. In [13] LTL with only eventually and globally operators (but without next and until operators) was considered for efficient translation to deterministic automata. The temporal logic specifications for reactive systems often consist of a set of assumptions and a set of guarantees, and the reactive system must satisfy the guarantees if the environment satisfies the assumptions. In [14] the GR1 (generalized reactivity 1) fragment of LTL was introduced where each assumption and guarantee is a liveness condition; and it has been shown that GR1 synthesis is very effective to automatically synthesize industrial protocols such as the AMBA protocol [15], [16].

Our contributions. In this work we consider the distributed synthesis problem for fragments of LTL. The previous results in literature considered the whole LTL and characterized architectures that lead to decidability of distributed synthesis. In contrast, we consider fragments of LTL to present finer characterizations of the decidability results. Our main contributions are as follows:

1) Reachability properties. First we consider the fragment of LTL with next and eventually (reachability) as the only temporal operators, and establish that the distributed synthesis problem is undecidable if there is an information fork in the underlying architecture. In particular, the problem is undecidable with one nesting depth of the next operator and only one eventually operator; i.e., if we consider the fragment of LTL that consists of Boolean combinations of atomic propositions and next of atomic propositions; and only one eventually as the temporal operator, then the distributed synthesis problem is undecidable iff there is an information fork in the architecture.

2) Safety properties. We then consider the fragment of LTL with next and globally (safety) as the only temporal operators, with a single occurrence of the globally operator. We show that the distributed synthesis problem can be decidable under the existence of information forks; in particular we establish decidability (in EXPSPACE) for the star architecture where processes have no common inputs from the environment. However, we show that
the problem remains undecidable for architectures containing an information fork-meet, a structure in which two processes receive sets of disjoint inputs, (as in the information fork case), and a third process receives the union of those sets. Moreover, our undecidability proof again uses specifications that do not contain nested next operators. In other words, if there is information fork, the problem may be decidable, but if there is information fork, and then the forked information meets again, then we obtain undecidability.

3) Temporal specifications without the next operator. Since our results show that even with one nesting depth of the next operator, distributed synthesis is undecidable with reachability and safety objectives, we finally consider the problem without the next operator. We show that if we consider a set of safety assumptions, and a set of guarantees such that each guarantee is a safety, reachability, or a liveness guarantee, then the distributed synthesis problem is decidable (and NEXPTIME-complete) for all architectures.

Hence, our paper improves upon existing results by presenting finer (un)decidability characterizations of the distributed synthesis problem for fragments of LTL. We also remark that when we establish decidability, it is either EXPSPACE or NEXPTIME-complete, as compared to previous proofs of decidability in distributed synthesis setting where the complexity is non-elementary. Thus as compared to the complexity of previous decidability results (tower of exponentials), our complexities (at most two exponentials) are very modest.

II. Model Description

Architectures. An architecture $A = (\mathcal{P}, p_e, V, E)$, where $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ is a set of $n + 1$ processes, $p_e$ is a distinguished process representing the environment, $V$ is a set of (output) binary variables, and $E: \mathcal{P} \times \mathcal{P} \rightarrow 2^V$ defines the communication variables between processes (i.e., $E(p, q) = \{u, v\}$ means that $p$ writes to variables $u$, $v$, and $q$ reads from them). For every process $p \in \mathcal{P}$, we denote with $O(p) = \bigcup_{q \in \mathcal{P}} E(p, q)$ the set of output variables of $p$, and with $I(p) = \bigcup_{q \in \mathcal{P}} E(q, p)$ the set of input variables of $p$. We require that for all $p, q \in \mathcal{P}$: $O(p) \cap O(q) = \emptyset$, i.e., no two processes write to the same variable. Finally, we will denote with $\mathcal{P}^- = \mathcal{P} \setminus \{p_e\}$.

An architecture describes a distributed reactive system, with the environment providing the inputs via $O(p_e)$, and the system responding via $I(p_e)$. The pair $(\mathcal{P}, E)$ describes the architecture of the system as a multigraph, with $\mathcal{P}$ being the set of nodes, and $E(p, q)$ the set of directed $p \rightarrow q$ edges with the corresponding variables as labels.

Trees. We define a (full) B-tree $T$ over some finite set $B$ as the set of all nodes $x \in (2^B)^*$. A (possibly infinite) sequence of nodes $\pi = (x_1, x_2, \ldots)$ forms a path in $T$, if for every $i \geq 1$ we have $x_{i+1} = x_i z$, for some $z \in 2^B$. For such a path $\pi$, we will use $\pi[i]$ to denote the element of $\pi$ in the $i$-th position, while $\pi[i, \infty]$ denotes the infinite suffix of $\pi$ starting at position $i$. An $A$-labeled B-tree $T_\lambda$ is a B-tree equipped with a labeling function of its nodes, $\lambda: (2^B)^* \rightarrow 2^A$. For every node $x = yz \in T_\lambda$ with $z \in 2^B$ we denote with $\ell_\lambda(x) = z \cup \lambda(x)$, i.e., the $\ell_\lambda$ of $x$ consists of the branch $z$ from the parent and the label $\lambda$ of $x$. For a (possibly infinite) path $\pi = (x_1, x_2, \ldots)$, we define with $\ell_\lambda(\pi) = (\ell_\lambda(x_1), \ell_\lambda(x_2), \ldots)$.

Local strategies. For every process $p \in \mathcal{P}^-$, a local strategy $\sigma_p$ is a function $\sigma_p: (2^{|\mathcal{P}|})^* \rightarrow 2^{O(I_p)}$, setting the output variables of $p$ according to the history of its input variables. A local strategy $\sigma_p$ has finite memory if there exists a finite set $M$, $m_0 \in M$, and functions $f: M \times 2^{|\mathcal{P}|} \rightarrow M$ and $g: M \rightarrow 2^{O(p)}$ such that for all $x = x_1 x_2 \ldots x_k$ with $x_i \in 2^{|\mathcal{P}|}$, we have $\sigma_p(x) = g(f(\ldots (f(f(m_0, x_1), x_2), \ldots), x_k))$. The memory of $\sigma_p$ is said to be $|M|$, while if $|M| = 1$, then $\sigma_p$ is called memoryless.

Collective strategies. Every such local strategy $\sigma_p$ can be viewed as a labeling of an $O(p)$-labeled $I(p)$-tree $T_{\sigma_p}$. The collective strategy of the architecture $A$ is a function $\sigma: (2^{|\mathcal{P}|})^* \rightarrow 2^{V\setminus O(p_e)}$, mapping every finite sequence of the outputs of the environment to a subset of the outputs of the processes $p$ according to the composition ($\sigma_p = p \in \mathcal{P}^-$). The collective strategy $\sigma$ can be viewed as a $(V \setminus O(p_e))$-labeled $O(p_e)$-tree $T_\sigma$ and for any infinite path $\pi$ in $T_\sigma$, we will call $\ell_\sigma(\pi)$ a computation. Hence, $T_\sigma$ describes a distributed algorithm, and every infinite path $\pi = (x_1, x_2, \ldots)$ starting from the root represents a distributed computation $\ell_\sigma(\pi)$, according to the local strategies ($\sigma_p: p \in \mathcal{P}^-$).

Synthesis (realizability). We will consider distributed reactive systems with specifications given by temporal logic formulas. For temporal logic formulas we will consider LTL; see [5] for the formal syntax and semantics of LTL. The problem of realizability of a temporal logic formula $\phi$ in an architecture $A$ asks whether there exist local strategies $\sigma_p$ for every process $p$, such that for every infinite path $\pi$ of the $(V \setminus O(p_e))$-labeled $O(p_e)$-tree $T_\sigma$ of the collective strategy $\sigma$, with $\pi$ starting from the root, we have $\ell_\sigma(\pi) \models \phi$. If $\phi$ admits such strategies $\sigma_p$ for every $p \in \mathcal{P}^-$, then it is called realizable, and the collective strategy $\sigma$ gives an implementation for $\phi$ in $A$.

III. Synthesis for Reachability Specifications

In the current section we discuss the synthesis problem for reachability specifications, where the objective consists of propositional formulas connected with Boolean operators and non-nested $\mathcal{X}$ (next) operators. We will show that even under such restrictions, the synthesis problem remains undecidable for all architectures containing an information fork, via a reduction from the halting problem of Turing machines.

Fragment LTL_$. We consider LTL_ which consists of formulas $\phi$ from the following LTL fragment:

$$
\begin{align*}
\theta & = P \mid \mathcal{X} P \\
\psi & = \theta_1 \wedge \theta_2 \mid \theta_1 \vee \theta_2 \mid \neg \theta \\
\phi & = Q \mid \diamond \psi
\end{align*}
$$

where $P, Q$ are propositional formulas, $\mathcal{X}$ is the next operator, $\diamond$ is the eventually temporal operator. We consider the standard semantics of LTL. Formula $\diamond \psi$ represents a reachability objective, and $Q$ will capture the initial input in the architecture.

Turing machines. Let $M$ be a deterministic Turing machine fixed throughout this section and let $Q$ be the set of states
of \( M \) (see [17] for detailed descriptions of Turing machines). The machine \( M \) works over the alphabet \( \{0, 1, \sqcup\} \), and its tape is bounded by \# symbol. The machine \( M \) cannot move left on a \# symbol, and moving right to a \# symbol effects in extending the tape by a blank symbol \( \sqcup \). In our analysis, \( M \) starts with the empty tape. A configuration of \( M \) is a word \#vquw\# \( q \), where \( a \in \{0, 1\} \), \( v, u \in \{0, 1\}^* \) and \( q \in Q \). Such a configuration has the standard interpretation as an infinite tape such that \( v \) is the part of the tape preceding the head, \( q \) is the current state of \( M \), \( a \) is the letter under the head, and \( u \) is a sequence of symbols succeeding the head. The blank symbol \( \sqcup \) represents the rightmost cell of the tape that has not been altered by \( M \). We define the projection \( \pi_\sqcup \) over words \( w \) from some alphabet containing \( \sqcup \), such that \( \pi_\sqcup(w) \) is the result of omitting all \( \sqcup \) symbols from \( w \). We define a scattered configuration \( C \) of \( M \) as a word over \( \Sigma = \{0, 1, \sqcup, \sqcap, \#\} \cup Q \) such that \( \pi_\sqcup(C) \) is a configuration of \( M \).

**Information-fork architecture.** We first consider the architecture \( A_0 \) (Figure 1), characterized as an information fork in [10], for which the problem of realizability has been shown to be undecidable, using LTL formulae with nested until operators (in [4]). Here we show that the problem remains undecidable for \( A_0 \) and specifications in the restricted fragment of LTL\( _0 \). This is obtained through a reduction from the halting problem of \( M \), by constructing a specification \( \phi \in \text{LTL}_0 \) which is realizable iff \( M \) halts on the empty input.

**Proof idea.** The architecture \( A_0 \) consists of the environment \( p_e \) and two processes \( p_1 \) and \( p_2 \). The processes act as I/O streams, outputting configurations of \( M \); the environment sends separately to each process next and stall signals, indicating that the corresponding process should output the next letter from \( \{0, 1, \sqcup, \#\} \cup Q \) of the current configuration of \( M \), or it should output \( \sqcap \).

**Construction of \( \varphi \).** First, we will provide a regular safety property \( \varphi \) which specifies that if the environment satisfies an alternation assumption, i.e., every stall signal is followed by a next signal, then \( p_1 \) and \( p_2 \) conform with a series of guarantees. The property \( \varphi \) does not belong to the LTL\( _0 \) fragment, but we will show how it can be expressed by a safety automaton \( A_{\text{safe}} \). Then, we will prove that if \( \varphi \) is realizable, and the environment conforms with the alternation assumption, then the processes output a legal sequence of configurations of \( M \), scattered with the \( \sqcap \) symbol.

**Conversion to LTL\( _0 \).** Next, we will provide the specification for the synthesis problem \( \phi \in \text{LTL}_0 \), such that \( \phi \) is realizable iff \( \varphi \) is realizable and \( M \) halts on the empty input. Formula \( \phi \) does not express \( \varphi \) directly, but it asserts that the environment simulates a run of \( A_{\text{safe}} \) faithfully, and finally one of the processes outputs a halting configuration of \( M \). More precisely, the environment simulates a run of \( A_{\text{safe}} \) storing the current state of \( A_{\text{safe}} \) in a set of hidden variables \( \{q_1, \ldots, q_m\} \in E(p_e, p_e) \), and \( \phi \) encodes that eventually either (i) the environment cheats in the simulation of \( A_{\text{safe}} \), or (ii) one of the processes outputs a halting state \( q \) of \( M \), while the current state of \( A_{\text{safe}} \) is not rejecting (i.e., \( q \) was reached legally with respect to \( M \)). We will conclude that \( \phi \) is realizable iff \( M \) halts on the empty input.

![Fig. 1: The architecture \( A_0 \) which consists an information fork.](image)

**Formal proof.** A safety automaton cannot express a scattered configuration that is finite. Thus, we define a scattered pre-configuration \( C \) of \( M \) as a (possibly infinite) word whose every finite prefix can be extended to a scattered configuration of \( M \). A scattered preconfiguration is formally defined as a finite or infinite word over \( \Sigma \) that begins with \#, there is at most one symbol from \( Q \), there are no symbols after the second \# and the \( \sqcup \) symbol is followed by the \# symbol.

Let \( C_1, C_2 \) be scattered preconfigurations. We denote with \( \sqcap(C) \) the set of positions in \( C \) where \( \sqcap \) occurs, and write \( C_1 \sqsupset C_2 \) if the symmetric difference of \( \sqcap(C_1) \) and \( \sqcap(C_2) \) has at most one element, i.e., \( \#(C_1) \sqcap\#(C_2) \leq 1 \). We define as \( C_1 \sqsupset C_2 \) if \( C_1 \sqsupset C_2 \) and

(i) \( \pi_\sqcap(C_2) \) follows legally from \( \pi_\sqcap(C_1) \) according to \( M \), or

(ii) both \( C_1, C_2 \) are infinite preconfigurations such that every finite prefix can be extended to finite preconfigurations \( C_1', C_2' \) such that \( \pi_\sqcap(C_2') \) follows legally from \( \pi_\sqcap(C_1') \).

For infinite words \( w_1, w_2 \), we define \( w_1 \sqotimes w_2 \) as a word over \( \Sigma \times \Sigma \) such that the \( i \)-th letter of \( w_1 \sqotimes w_2 \) is a pair of the \( i \)-th letters of \( w_1, w_2 \). Observe that there are safety automata working over \( \Sigma \times \Sigma \) that recognize the languages \( \{C_1 \sqotimes C_2 : C_1 \sqsupset C_2\} \) and \( \{C_1 \sqotimes C_2 : C_1 \sqsupset C_2\} \).

**Construction of \( \varphi \).** We first construct the regular safety property \( \varphi = L \rightarrow \bigwedge_{0 \leq i \leq 4} \text{Cond}_i \), where \( L \) (the alternation assumption) and \( \text{Cond}_i \) are defined as follows:

- \( L \): for every process, every stall signal is followed by a next signal.
- \( \text{Cond}_0 \): each process outputs \( \sqcap \) when its input is \( \sqcap \), otherwise it outputs a letter from \( \Sigma \setminus \{\sqcap\} \),
- \( \text{Cond}_1 \): each process produces a sequence of scattered preconfigurations,
- \( \text{Cond}_2 \): initially, each process produces two scattered configurations of \( M \), whose projections are the first two valid configurations of \( M \),
- \( \text{Cond}_3 \): if starting from some position \( i, p_1 \) outputs consecutively \( C_1, C_2 \) and \( p_2 \) outputs consecutively \( C_1', C_2' \), then \( C_1' \sqsupset C_1 \) implies \( C_2' \sqsupset C_2 \) or \( C_2' \sqsupset C_2 \),
- \( \text{Cond}_4 \): if \( D, D' \) are outputs of \( p_1, p_2 \) up to some positions such that \( D \sqsupset D' \) and \( |\pi_\sqcap(D)| = |\pi_\sqcap(D')| \), then \( \pi_\sqcap(D) = \pi_\sqcap(D') \).

We provide a high-level description of the construction of an alternating safety automaton \( A_{\text{safe}} \) (see [18] for the definition of alternating automata) which verifies that every execution satisfies \( \varphi \). Note that \( A_{\text{safe}} \) can be transformed to a non-deterministic automaton by a standard power-set construction. Clearly, conditions \( L, \text{Cond}_0 \) and \( \text{Cond}_1 \) can be expressed...
by a safety automaton. For the condition Cond₂, observe that the first two configurations of M have at most 9 letters \#q₀ \land \#q₁ a \land \#, with a \in \{0, 1, \epsilon\}. To show that the rest of conditions can be expressed by a safety automaton, we assume that \mathcal{L} is satisfied; otherwise those conditions do not have to be checked (note that if \mathcal{L} is violated, \mathcal{A}_{safe} accepts unconditionally). Because of \mathcal{L}, \mathcal{A}_{safe} can verify that p₁ and p₂ conform with Cond₂ by checking the first 18 output letters. For the condition Cond₃, \mathcal{A}_{safe} operates as follows: whenever it encounters a \# symbol marking the beginning of a configuration, it splits universally. One copy looks for the next configuration, and the second copy, denoted by \mathcal{A}_3, verifies that Cond₃ holds at the current position, as follows. It ignores \perp symbols and compares whether C₁ \parallel C₁′, configurations \piₙ(C₁) and \piₙ(C₁′) are equal everywhere except for positions adjacent to the head of M, and the letters adjacent to the head are consistent with the transition of M. If one of these conditions is violated, C₁′ \not\parallel C₁, therefore \mathcal{A}_3 accepts the word regardless of what follows. Otherwise, if those conditions hold, i.e., C₁′ \parallel C₁, \mathcal{A}_3 non-deterministically verifies one of the following conditions: C₂′ \not\parallel C₂ or C₂′ \parallel C₂. Both conditions can be verified by safety automata, since C₂ and C₂′ either start concurrently, or C₂ is delayed by 1 step from C₂′. For the condition Cond₄ observe that if D \parallel D′ and ||\piₙ(D) = \piₙ(D′)||, then ||D−D'|| \leq 1 and the automaton needs to remember at most one symbol to compare \piₙ(D) and \piₙ(D′). We can now prove the following lemma.

**Lemma 1.** If \varphi is realizable, then for every k \in \mathbb{N}, in all executions where \mathcal{L} holds, both p₁ and p₂ output sequences of scattered configurations whose \piₙ projections are sequences of at least k consecutive valid configurations of M, starting with the initial configuration on the empty input.

**Proof:** First note that there exist executions where the environment indeed satisfies \mathcal{L}, and thus p₁ and p₂ satisfy conditions Cond₀-Cond₄. The lemma clearly holds for k = 1, 2, due to conditions Cond₀ = Cond₂. For the inductive step, assume that the lemma holds for k \geq 2. Consider a sequence of inputs to p₁ consisting of next signals only. Then, there is a sequence of inputs to p₂ consisting of some number of next signals and exactly |\piₙ(Cₖ)| stall signals placed in such a way that p₁ outputs C₁ \ldots CₖCₖ₊₁, p₂ outputs C₁′ \ldots Cₖ₋₁Cₖ′, and CₖCₖ₊₁, Cₖ₋₁Cₖ′ are synchronized, i.e., they start at the same position and Cₖ \parallel Cₖ′ \parallel Cₖ₊₁ \parallel Cₖ′. By the induction assumption \piₙ(Cₖ₊₁) and \piₙ(Cₖ′) = \piₙ(Cₖ) are, respectively, (k−1)-th and k-th configurations of M. Therefore, Cₖ₊₁ \parallel Cₖ and, by Cond₃, Cₖ′ \parallel Cₖ₊₁. This implies that Cₖ₊₁ is a finite scattered preconfiguration and \piₙ(Cₖ₊₁) is the (k+1)-th configuration of M.

Given that for an input consisting of next signals only, p₁ outputs C₁ \ldots CₖCₖ₊₁ satisfying the statement, we can show that regardless of the number of stall signals, under condition \mathcal{L}, p₁, p₂ output k + 1 scattered configurations satisfying the statement. First, the condition Cond₄ implies that if p₂ also has an input sequence consisting of next signals alone, it will output the same sequence, that is, C₁ \ldots CₖCₖ₊₁. By a simple induction on the number of stall signals each process receives, and condition Cond₄, we conclude that for any number of stall signals, as long as \mathcal{L} is satisfied by the environment, p₁, p₂ output k + 1 scattered configurations whose projections are the first k + 1 consecutive configurations of M.

**Conversion to LTL.** Given the safety automaton \mathcal{A}_{safe} which verifies that \varphi is satisfied, we can construct a specification \varphi \in LTL, such that \varphi is realizable if and only if the Turing machine M does not halt on the empty input. The environment uses the hidden (not visible to p₁, p₂) variables q₁, \ldots, qₖ \in E(pₑ, pₑ) to simulate the automaton \mathcal{A}_{safe}. We provide a high level description of the following formulas:

Q specifies that the first state of \mathcal{A}_{safe} according to the output variables \{q₁, \ldots, qₖ\} is compatible with the initial values of x₁, x₂, y₁ and y₂ (i.e., \{q₁, \ldots, qₖ\} represent the state of \mathcal{A}_{safe} reached from the initial state after reading the initial values of x₁, x₂, y₁ and y₂; Q is propositional)

ψ₁ specifies that \mathcal{A}_{safe} has a transition from the current state to the next state, encoded by the values of \{q₁, \ldots, qₖ\} in the current and the next round, according to the value of variables x₁, x₂, y₁ and y₂ in the next round (i.e., pₑ simulates \mathcal{A}_{safe} faithfully; ψ₁ contains only propositions and non-nested \mathcal{X} operators).

ψ₂ specifies that the current state of \mathcal{A}_{safe} is not rejecting, and p₁ or p₂ output a halting state of M (i.e., a halting configuration of M was reached by some process, and both processes behaved according to \mathcal{A}_{safe}; ψ₂ is propositional).

Finally, we construct \phi = Q \land ψ₁ \land ψ₂, with \phi \in LTL. If \phi is realizable, the processes satisfy ψ₂ in all runs where the environment faithfully simulates \mathcal{A}_{safe} and conforms with condition \mathcal{L}(i.e., Q and ψ₁ are true). Then p₁, p₂ output a halting state of M and satisfy \varphi, which by Lemma 1, guarantees that the halting state was reached by a legal sequence of configurations of M. In the inverse direction, if M halts, then \phi is realizable by (finite) local strategies which output a finite, legal sequence of configurations of M and conform with condition Cond₀. Hence, we obtain the following theorem.

**Theorem 1.** The realizability of specifications from LTL₀ in \mathcal{A} is undecidable.

Similarly as in [10], the above argument can be carried out to any architecture which contains an information fork, by introducing additional safety conditions in \varphi, which require that all processes propagate the inputs of the environment to the two processes constituting the information fork. It has also been shown in [10] that in architectures without information forks, the realizability of every LTL specification is decidable. Hence, Theorem 1 together with the results from [10] lead to the following corollary.

**Corollary 1.** For every architecture \mathcal{A}, the realizability of specifications from LTL₀ in \mathcal{A} is decidable if \mathcal{A} does not contain an information fork.

IV. SYNTHESIS FOR SAFETY SPECIFICATIONS

In the current section we consider safety specifications where the safety condition consists of propositional formulas
connected with Boolean operators, and the $\land$ temporal operator. First, we show that the synthesis problem is undecidable for architectures containing an information fork-meet (see Figure 3), by a similar construction as in the case of $\text{LTL}_{\land}$. Then we show that the problem is decidable for the family of star architectures, despite the existence of information forks.

**Fragment LTL$_\land$.** We consider LTL$_\land$ that consists of formulas $\phi$ from the following LTL fragment:

$$
\begin{align*}
\psi &= P \land \psi_1 \land \psi_2 \land \psi_3 \land \neg \psi \land X \psi \\
\phi &= Q \land \Box \psi
\end{align*}
$$

where $P$, $Q$ are propositional formulas, and $\Box$ is the globally operator. We consider the standard semantics of LTL. The $\Box \psi$ part of $\phi$ specifies a safety condition, and we interpret the $Q$ part as the initial conditions. The fragment LTL$_\land$ can express safety specifications, one of the most basic specification in verification.

While the information fork criterion is decisive for the undecidability of reachability specifications, here we extend this criterion to the family of star architectures of $n + 1$ processes, denoted as $S_n$ (i.e., $p_e$ is the central process, and $\bigcup I(p_i) = O(p_e)$ (Figure 2) and show that: (i) the realizability of some $\phi \in \text{LTL}_\land$ in $S_n$ is decidable if all processes receive pairwise disjoint inputs, (ii) it is undecidable if $n \geq 3$ and we allow overlapping inputs. The latter can be generalized to all architectures which contain such a structure, which we call an information fork-meet.

**A. Overlapping inputs**

Here we demonstrate undecidability of realizability of specifications $\phi \in \text{LTL}_\land$ for star architectures with overlapping inputs, and with $\phi$ having $X$-depth 1 (i.e., $\phi$ belongs to a subclass of LTL$_\land$ where $X$ operators are not nested). We first consider the star architecture $A_1$ (Figure 3), and obtain the undecidability of realizability of such specifications via a reduction from the (non) halting problem.

Given a Turing machine $M$, recall the specification $\varphi$ (from Section 3 for $\text{LTL}_{\land}$) encoding conditions $L$ and $\text{Cond}_0 - \text{Cond}_4$ through the safety automaton $A_{\text{safe}}$. In contrast with the previous section, here we require that process $p_3$ (instead of $p_e$) faithfully simulates the safety automaton $A_{\text{safe}}$ using the output variables $q_1, \ldots, q_m \in E(p_3, p_e)$. Note that $A_{\text{safe}}$ operates on the variables $x_1, x_2, y_1, y_2$, while $p_3$ does not have access to $y_1$ and $y_2$. However, it can infer these values by simulating $p_1$ and $p_2$ internally, since $p_3$ receives both $x_1$ and $x_2$ (overlapping inputs).

![Fig. 2: The family of start architectures $S_n$.](image)

![Fig. 3: The architecture $A_1$ consists an information fork-meet.](image)

**Formal proof.** We provide a high level description of the following formulas:

$Q$ specifies that the first state of $A_{\text{safe}}$ according to the output variables $\{q_1, \ldots, q_m\}$ is compatible with the initial values of $x_1$, $x_2$, $y_1$ and $y_2$ (i.e., $\{q_1, \ldots, q_m\}$ represent the state of $A_{\text{safe}}$ reached from the initial state after reading the initial values of $x_1$, $x_2$, $y_1$ and $y_2$; $Q$ is propositional)

$\psi_1$ specifies that $A_{\text{safe}}$ has a transition from the current state to the next state, encoded by the values of $\{q_1, \ldots, q_m\}$ in the current and the next round, according to the value of variables $x_1$, $x_2$, $y_1$ and $y_2$ in the next round (i.e., $p_e$ simulates $A_{\text{safe}}$ faithfully; $\psi_1$ contains only propositional and non-nested $X$ operators).

$\psi_2$ specifies that $p_1$ and $p_2$ do not output a halting state of $M$ (i.e., $M$ does not terminate; $\psi_2$ is propositional).

$\psi_3$ specifies that $A_{\text{safe}}$ does not reach a rejecting state (i.e., the processes conform to conditions $\text{Cond}_0 - \text{Cond}_4$ or the environment violates $L$; $\psi_3$ is propositional).

We construct $\phi = Q \land \Box(\psi_1 \land \psi_2 \land \psi_3)$. Similarly as in the case of $\text{LTL}_{\land}$, if $\phi$ is realizable, $p_3$ faithfully simulates $A_{\text{safe}}$ ($Q$ and $\psi_1$ are true), and $p_1$, $p_2$ satisfy $\varphi$ in all runs where the environment conforms with condition $L$ ($\psi_3$ is true). By Lemma 1, $p_1$, $p_2$ output a legal sequence of configurations of $M$, and $\psi_2$ guarantees that $M$ does not halt. In the inverse direction, if $M$ does not halt, then $\phi$ is realizable by local strategies for which (i) $p_1$, $p_2$ output a legal sequence of configurations of $M$ and conform with condition $\text{Cond}_0$, and (ii) $p_3$ faithfully simulates $A_{\text{safe}}$. Hence we have the following result.

**Theorem 2.** The realizability of specifications from LTL$_\land$ in $A_1$ is undecidable.

**Remark 1.** We remark that our proof of undecidability in Theorem 2 makes use of infinite-memory strategies, since the processes $p_1$ and $p_2$ are required to output an infinite, non-halting computation. However, we show that even if we restrict the problem to finite-memory strategies, the realizability problem for LTL$_\land$ in $A_1$ still remains undecidable.

**Undecidability for finite-memory strategies.** We define the looping problem as follows. Given a deterministic Turning Machine $M$, decide whether $M$, started on the empty tape, uses only finite memory and does not terminate. The looping problem is undecidable. Indeed, consider the following reduction from the halting problem to the looping problem. For a Turing Machine $N$, we construct Turing Machine $\tilde{N}$ that simulates $N$ and counts the number of steps taken by $N$. This
counter is stored on the tape of $\tilde{N}$. If $N$ halts, $\tilde{N}$ loops at the last configuration and does not increase the counter. Clearly, $N$ started on the empty tape halts iff $\tilde{N}$ uses only finite memory and does not terminate.

Observe that the reduction in Theorem 2 restricted to finite-memory strategies yields a reduction from the looping problem to the finite-memory realizability problem of LTL$\Box$ in the architecture $A_1$.

Indeed, recall that for a (deterministic) Turing Machine $M$, there are strategies $\sigma_1, \sigma_2$ that realize the specification $\phi$ if $M$ does not halt. Let $C_0, C_1, \ldots$ be a sequence of configurations of $M$ started at the empty input. Observe that $M$ loops started on the empty tape iff $C_0, C_1, \ldots$ are of bounded length. It remains to be shown that $C_0, C_1, \ldots$ are of bounded length iff $\sigma_1, \sigma_2$ are finite memory.

Assume that $C_0, C_1, \ldots$ are of bounded length. Since $M$ is deterministic, the infinite sequence of configurations is of the form $C_1 \ldots C_k (C_{k+1} \ldots C_m)\omega$, for some $k, m$ with $k < m$. It follows that there are finite-memory strategies that realize the specification $\phi$. Conversely, assume that $\sigma_1, \sigma_2$ are finite-memory strategies realizing the specification $\phi$. Consider a run where the environment gives only next signals to the process $p_1$. Since the strategies are deterministic, the length of configuration is bounded by the size of the memory of $\sigma_1$ and $\sigma_2$, or it is infinite because a strategy loops. As $\sigma_1, \sigma_2$ output infinitely many configurations, all the configurations $C_0, C_1, \ldots$ are finite, and thus they are of length bounded by the memory of $\sigma_1$ and $\sigma_2$.

Information fork-meet. We say that an architecture $A = (P, p_e, V, E)$ has an information fork-meet if there are three processes $p_1, p_2, p_3 \in P \setminus \{p_e\}$ and paths $\pi_1, \pi_2$ in the underlying graph such that

1. the first edges in $\pi_1, \pi_2$ are labeled by output variables of $p_e$,
2. the last edge of $\pi_1$ is an input variable of $p_1$, but not $p_2$,
3. the last edge of $\pi_2$ is an input variable of $p_2$, but not $p_1$,
4. the last edges of $\pi_1, \pi_2$ are input variables of $p_3$.

Observe that an information fork-meet is a special case of information fork, with a third process that collects all information that is divided between $p_1$ and $p_2$.

As in the case of LTL$\Box$, the undecidability argument can be carried to any architecture containing such a structure, by introducing additional conditions in $\varphi$ which require the rest of the processes to propagate the inputs of the environment to $p_1, p_2$ and $p_3$ accordingly.

Corollary 2. The realizability of specifications from LTL$\Box$ in architectures containing an information fork-meet is undecidable.

B. Pairwise disjoint inputs

In this subsection we discuss synthesis for formulas $\phi \in$ LTL$\Box$ for the class of star architectures, with the additional property that all pairs of processes receive disjoint inputs (i.e., $\forall i \neq j : I(p_i) \cap I(p_j) = \emptyset$), denoted as $\mathcal{S}_n$. Our goal is to prove decidability of realizability of such $\phi \in$ LTL$\Box$ in every architecture $A \in \mathcal{S}_n$, by showing that whenever such $\phi$ is realizable, it admits strategies of bounded memory.

Consider some architecture $A \in \mathcal{S}_n$, and an arbitrary $\phi = \Box \psi \in$ LTL$\Box$, with the nesting level of $\psi$ being $k$. Assume that $\phi$ is realizable in $A$ by local strategies $\sigma_i$ for every process $p_i$. These strategies can be represented by $O(p_i)$-labeled $I(p_i)$-trees $T_{\sigma_i}$. We will show how to construct strategies $\tau_i$ that also realize $\phi$, where each tree $I(p_i)$-tree $T_{\tau_i}$ representing $\tau_i$ is defined from first $2^{2l(p_i)}+1$ levels of $T_{\sigma_i}$, by applying a folding function given below. We first define the notion of some $i \in N$ closing $-\psi$.

Definition 1. For a computation $l(\pi)$ and some $i \in N$ we say that $i$ closes $-\psi$ in $l(\pi)$ if $l(\pi)[i - k, \infty] \models -\psi$.

Remark 2. $l(\pi) \models \phi$ iff no $i$ closes $-\psi$ in $l(\pi)$.

Let $\sigma_1, \ldots, \sigma_n$ be local strategies and $\sigma$ be the collective strategy induced by $\sigma_1, \ldots, \sigma_n$. For every $i \in \{1, \ldots, n\}$, the local strategy $\sigma_i$ is represented by an $O(p_i)$-labeled $I(p_i)$-tree $T_{\sigma_i}$. For every node $x \in T_{\sigma_i}$, with $|x| \geq k$, we denote with $\pi_x = (x_k, x_{k-1}, \ldots, x_1)$ the $k$-node suffix of the unique path to $x = x_1$, and define the type of $x$ under $\sigma_i$ as $t_{\sigma_i}(x) = \ell_{\sigma_i}(\pi_x)$. For every level $l \geq k$ we define the type of $l$ under $\sigma$ as $t_{\sigma}(l) = \{t_{\sigma_i}(x) : i \in \{1, \ldots, n\}, x \in T_{\sigma_i}, |x| = l\}$, i.e., the type of a level $l$ is the set of the types of the nodes of level $l$ every $T_{\sigma_i}$, where $i \in \{1, \ldots, n\}$. Note that there exist at most $2^{k|V|}$ distinct types of nodes. Consequently, there exist at most $2^{k|V|}$ distinct types of levels.

Consider some architecture $A \in \mathcal{S}_n$ and an arbitrary $\phi = \Box \psi \in$ LTL$\Box$, with the nesting level of $X$ operators in $\psi$ being $k$. Assume that $\phi$ is realizable in $A$ by local strategies $\sigma_i$ for every process $p_i$. These strategies can be represented by $O(p_i)$-labeled $I(p_i)$-trees $T_{\sigma_i}$. We will show how to construct strategies $\tau_i$ that also realize $\phi$, where each tree $I(p_i)$-tree $T_{\tau_i}$ representing $\tau_i$ is defined from first $2^{2l(p_i)}+1$ levels of $T_{\sigma_i}$, by applying a folding function given below. We first define the notion of some $i \in N$ closing $-\psi$.

Definition 1. For a computation $l(\pi)$ and some $i \in N$ we say that $i$ closes $-\psi$ in $l(\pi)$ if $l(\pi)[i - k, \infty] \models -\psi$.

Remark 2. $l(\pi) \models \phi$ iff no $i$ closes $-\psi$ in $l(\pi)$.

Let $\sigma_1, \ldots, \sigma_n$ be local strategies and $\sigma$ be the collective strategy induced by $\sigma_1, \ldots, \sigma_n$. For every $i \in \{1, \ldots, n\}$, the local strategy $\sigma_i$ is represented by an $O(p_i)$-labeled $I(p_i)$-tree $T_{\sigma_i}$. For every node $x \in T_{\sigma_i}$, with $|x| \geq k$, we denote with $\pi_x = (x_k, x_{k-1}, \ldots, x_1)$ the $k$-node suffix of the unique path to $x = x_1$, and define the type of $x$ under $\sigma_i$ as $t_{\sigma_i}(x) = \ell_{\sigma_i}(\pi_x)$. For every level $l \geq k$ we define the type of $l$ under $\sigma$ as $t_{\sigma}(l) = \{t_{\sigma_i}(x) : i \in \{1, \ldots, n\}, x \in T_{\sigma_i}, |x| = l\}$, i.e., the type of a level $l$ is the set of the types of the nodes of level $l$ every $T_{\sigma_i}$, where $i \in \{1, \ldots, n\}$. Note that there exist at most $2^{k|V|}$ distinct types of nodes. Consequently, there exist at most $2^{k|V|}$ distinct types of levels.

Folding function. Assume that there exist two levels $l_1 < l_2$ such that $t_{\sigma}(l_1) = t_{\sigma}(l_2)$. Then for every tree $T_{\sigma_i}$, for every node $x$ in level $l_2$ there exists a node $y$ in level $l_1$ such that $t_{\sigma_i}(x) = t_{\sigma_i}(y)$, i.e., $x$ and $y$ have the same type. For such $l_1, l_2$, and every process $p_i$, we define the folding function $f_i : (2^{l(p_i)})^* \rightarrow (2^{l(p_i)})^*$ recursively as follows:

$$f_i(x) = \begin{cases} x & \text{if } |x| < l_2 \\ y & \text{if } |x| = l_2 \text{ where } |y| = l_1 \text{ and } t_{\sigma_i}(x) = t_{\sigma_i}(y) \\ f_i(f_i(y)z) & \text{if } |x| > l_2 \text{ for } x = yz \text{ with } z \in 2^{l(p_i)} \end{cases}$$

and construct local strategies $\tau_i(x) = \sigma_i(f_i(x))$. Hence, every strategy $\tau_i$ behaves as $\sigma_i$, up to level $l_2$, while for nodes further below, it maps them to nodes between levels $l_1$ and $l_2$, by recursively folding the levels $l_1$ and $l_2$ with respect to the types of their nodes.

The strategies $\tau_i$ have the property that they preserve the types of all local nodes up to level $l_2$, and only those. Because of the pairwise disjoint inputs, this property is implied for the global nodes of the collective strategy $\tau$ as well. Observe that the set of all such types serves as the set $\Delta$ of Remark 3,
which in turn guarantees that the collective strategy \( \tau \) also realizes \( \phi \). We formalize these arguments below.

The following lemma establishes that for all nodes \( x \) in all \( T_\sigma \), the type of \( x \) is the same as the type of its image under \( f_i \) in the corresponding \( T_\sigma \).

**Lemma 2.** For every \( x \in (2^{I(p)})^* \) with \(|x| \geq k \), we have that \( t_\tau(x) = t_\sigma(f_i(x)) \).

**Proof:** Our proof proceeds by induction on \(|x|\):
1. \(|x| < l_2\): For all nodes \( w \) in \( \pi_x \), we have that \( t_\tau(w) = \sigma_i(f_i(w)) = \sigma_i(w) \), hence \( t_\tau(x) = t_\sigma(f_i(x)) \).
2. \(|x| = l_2\): The statement holds by definition.
3. \(|x| = m + 1\): Let \( x = yz \) with \(|y| = m \). By the inductive hypothesis, \( t_\tau(y) = t_\sigma(f_i(y)) \). We distinguish between the following cases, depending on whether \( f_i(y) \) extended by \( z \) hits the level \( l_2 \) (Figure 4):
   (i) \(|f_i(y)| < l_2 - 1\): Then \( f_i(x) = f_i(f_i(y))z = f_i(y).z \), that is, if we reach node \( x \) by extending node \( y \) by an edge \( z \), the same holds for their corresponding images under \( f_i \). Then \( t_\tau(x) = \sigma_i(f_i(y))z = \sigma_i(y)z \), thus \( t_\tau(x) = t_\sigma(f_i(y))z = t_\sigma(f_i(x)) \) (i.e., the strategy \( \tau_i \) will have \( x \) as \( \sigma_i \) labels its image \( f_i(x) \), thus the types of these two nodes are equal).
   (ii) \(|f_i(y)| = l_2 - 1\): By construction, \( t_\tau(f_i(x)) = t_\sigma(f_i(y))z \) (i.e., \( f_i(y) \) extended by \( z \) hits level \( l_2 \), and \( f_i \) is a node of the same type). Then \( t_\tau(x) = \sigma_i(f_i(x))z = \sigma_i(y)z \), hence as in (i), \( t_\tau(x) = t_\sigma(f_i(y))z = t_\sigma(x) \).

The desired result follows.

It follows from the above remark and Lemma 2, that for every \( x \in T_\sigma \) we have that \( t_\tau(x) = t_\sigma(f_i(x)) \), where \( f_i(x) = \bigcup f_i(x) \). That is, the local folding functions \( f_i \) result in a unique, global folding function \( f \), and the types in the corresponding collective strategy tree are preserved between the global nodes, and their images under \( f \). This implies that the set of types occurring in \( T_\sigma \) is a subset of types of \( T_\sigma \).

Then, by Remark 3 we conclude:

**Lemma 3.** The collective strategy \( \tau \) implements \( \phi \).

Hence, whenever for a realizable \( \phi \in \text{LTL}_{\Box} \) exist levels \( l_1 \) and \( l_2 \) with the same type under \( \sigma_i \), we can construct a collective strategy \( \tau \) for which every local strategy \( \tau_i \) uses only the first \( l_1 \) levels of the corresponding \( \sigma_i \), and Lemma 3 guarantees that \( \tau \) implements \( \phi \). By our previous observation and the pigeonhole principle, \( l_2 \) is upper bounded by \( 2^{k(V)} + 1 \), and thus every local strategy \( \tau_i \) operates in the first \( 2^{k(V)} + 1 \) levels of the corresponding \( I(p_i) \)-tree. There are a bounded number of local strategies with this property, thus the problem of realizability in this case reduces to exhaustively exploring all of them. Moreover, it follows from our analysis that local nodes in the same level and having the same type can be merged, since the local strategy that behaves identically in both subtrees preserves the set of types appearing in the global tree. Hence, the width of each level is bounded by the number of different possible types, \( 2^{k(V)} \).

**Theorem 3.** The realizability of \( \phi \in \text{LTL}_{\Box} \) for the class \( S_n \) of star architectures with pairwise disjoint inputs is decidable in \( \text{EXPSPACE} \).

**Proof:** We describe a non-deterministic Turing machine \( N \) which, given an architecture \( A \in S_n \) and a specification \( \phi \in \text{LTL}_{\Box} \) as input, decides whether there exist local strategies \( \sigma_i \) which realize \( \phi \). The machine \( N \) operates in levels, as follows: It first guesses the top \( k \) levels of each strategy \( \sigma_i \), it verifies that \( \phi \) has not been violated, and it computes the set \( \Delta_i \) of all types of nodes in the \( k \)-th level of each \( \sigma_i \). Then, iteratively for each level \( j \) up to \( 2^{k(V)} + 1 \), it guesses the sets \( \Delta_i^j \) of types of nodes of the new level of each \( \sigma_i \), such that \( \Delta_i \) is compatible with \( \Delta_i^{j-1} \), and verifies that all possible combinations of types between all \( \Delta_i \) do not violate \( \phi \).

It is straightforward to verify that if \( N \) accepts, \( \phi \) is realizable by local strategies \( \sigma_i \), such that for every \( x \in T_\sigma \), the strategy \( \sigma_i \) assigns to \( x \) the value of a type in \( \Delta_i^{k(x)} \) which is compatible with \( x \). Conversely, if there are local strategies \( \sigma_i \) that realize \( \phi \), the machine \( N \) can guess their levels, and thus accept. Therefore, if \( N \) rejects, no such strategies exist. Finally, \( N \) operates in \( \text{NEXPSPACE} \), since \( |\Delta_i| \leq 2^{k(V)} \), and the first \( k \) levels of each \( \sigma_i \) have less than \( 2^{k(V)} \) nodes. By Savitch’s Theorem [19], \( \text{NEXPSPACE} = \text{EXPSPACE} \).

V. SYNTHESIS WITHOUT THE NEXT OPERATOR

In the current section we consider a fragment of \( \text{LTL} \) without the \( X \) operator, for which the problem of realizability is decidable in non-deterministic exponential time in the size of the specification.
Fragment LTL\textsubscript{AG}. We consider LTL\textsubscript{AG} that consists of formulas $\phi$ from the following LTL fragment:

$$\phi = \bigwedge_i \Box P_i \rightarrow \left( \bigwedge_i \Box Q_i \land \bigwedge_i \Box R_i \land \bigwedge_i \Box F_i \right)$$

$$\equiv \Box \bigwedge_i P_i \rightarrow \left( \bigwedge_i \Box Q_i \land \bigwedge_i \Box R_i \land \bigwedge_i \Box F_i \right)$$

$$\equiv \Box P \rightarrow \left( \Box Q \land \bigwedge_i \Box R_i \land \bigwedge_i \Box F_i \right)$$

for $i \in \{1, \ldots, m\}$, with $P_i, Q_i, R_i, F_i$ propositional formulas, and $P = \bigwedge_i P_i$, $Q = \bigwedge_i Q_i$. We consider the standard semantics of LTL. We first observe that every realizable safety formula $\psi = \Box Q$, where $Q$ is propositional, admits memoryless strategies. The LTL\textsubscript{AG} can express specification that consists of conjunction of safety assumptions, and guarantees where each guarantee is a safety, reachability, or a liveness condition.

**Lemma 4.** Let $A$ be any architecture. Every formula $\psi = \Box Q$, for some propositional $Q$, is realizable in $A$ iff it is realizable by memoryless strategies.

**Proof:** The right to left direction is immediate. For the left to right direction, assume that $\psi$ is realizable, and let $\sigma_i$ be the strategy of process $p_i \in P$ in $A$. Construct strategies $\tau_i$ such that $\tau_i(x) = \sigma_i(z)$ for every $x = yz \in (2^I(p_i))^i$ and $z \in 2^{F(p_i)}$, and let $\tau$ be the collective strategy of $\tau_i$. For every infinite path in $T$, we have that $\ell_i(\tau)[i] = \ell_i(\sigma)[i]$ for all $i \in N$. Since $\sigma$ implements $\psi$, it follows that $\ell_i(\tau)[i] = Q$. Hence for all $i$, we have $\ell_i(\tau)[i] = Q$, and thus $\ell(\tau) = \psi$, and all $t$ are memoryless strategies.

The following lemma establishes that reachability and safety specifications of propositional formulas are equivalent with respect to realizability.

**Lemma 5.** Let $A$ be any architecture. For every formula $\psi = \Box Q$ for some propositional $Q$, $\psi$ is realizable in $A$ iff $\psi' = \Box Q$ is realizable in $A$.

**Proof:** The left to right direction is immediate. For the right to left direction, assume that there exist local strategies $\sigma_i$ such that the collective strategy $\sigma$ implements $\psi'$, and let $T_\sigma$ be the corresponding collective strategy tree. Then there exists some node $x \in T_\sigma$ such that for all $z \in 2^{O(p_i)}$, we have $\ell(x) = Q$, otherwise we can construct an infinite path $\pi$ such that $\pi[i] \not\equiv Q$ for all $i \in N$ (and thus $\ell(\pi) \not\equiv \psi'$), by choosing for every $x \in T_\sigma$ a $z \in 2^{O(p_i)}$ such that $\ell(x) \not\equiv Q$. It follows that for local strategies $\tau_i$ such that $\tau_i(yz) = \sigma_i(xz)$ for all $y, z$, the collective strategy $\tau$ realizes $\psi$.

Lemma 6 shows that the realizability of some $\phi \in \text{LTL}_{AG}$ reduces to realizing a set of safety formulas of the form of Lemma 4.

**Lemma 6.** Let $A$ be any architecture. Every formula $\phi \in \text{LTL}_{AG}$ is realizable in $A$ iff every $\phi_{R_i} = \Box (P \rightarrow (Q \land R_i))$ and every $\phi_{F_i} = \Box (P \rightarrow (Q \land F_i))$ is realizable in $A$.

**Proof:** (i) For the right to left direction, assume that there exist families of memoryless (by Lemma 4) local strategies ($\sigma_j^{R_i}$) and ($\sigma_j^{F_i}$) for every process $p_j$, such that the collective strategy $\sigma_j^{R_i}$ implements $\phi_{R_i}$, and the collective strategy $\sigma_j^{F_i}$ implements $\phi_{F_i}$. Construct local strategies $\tau_j$ such that for every $x = yz$ with $|z| = (1 + |x|) \mod 2m$, we have $\tau_j(x) = \sigma_j^{R_i}(z)$ if $|z| \leq m$, and $\tau_j(x) = \sigma_j^{F_i}(y)$ if $|z| > m$ (i.e. the local strategy $\tau_j$ repeatedly alternates between all the strategies $\sigma_j^{R_i}$ in the first $m$ steps, and between all the strategies $\sigma_j^{F_i}$ the next $m$ steps). Let $\tau$ be the collective strategy of all $\tau_j$ and consider an arbitrary path $\pi$ in $T$. Either $\ell_j(\tau)[|k|] = P$ for some $k$, or for all $k$, it holds $\ell_j(\tau)[|k|] = P$, and by construction, for $i = 1 + k \mod 2m$, we have $\ell_j(\tau)[|k|] = Q \land R_i$ when $i \leq m$ and $\ell_j(\tau)[|k|] = Q \land F_{i-m}$ when $i > m$. In both cases, $\ell_j(\pi) = \phi$.

(ii) For the left to right direction, assume that for some $i$, $\phi_{R_i}$ is not realizable (the analysis is similar for $\phi_{F_i}$). By Lemma 5, $\Box (P \rightarrow (Q \land R_i))$ is not realizable. Hence, for any collective strategy $\sigma$ there exists some path $\pi$ in $T$, such that for all $k$, we have $\ell_\sigma(\pi)[|k|] = P \land (\neg Q \lor \neg R_i)$, and $\sigma$ does not implement $\phi$.

Hence, Lemma 6 establishes that every formula $\phi \in \text{LTL}_{AG}$ is realizable if and only if it admits local strategies for all the corresponding $\phi_i$, by providing a constructive argument.

As a consequence of Lemma 4, deciding whether every $\phi_i$ is realizable reduces to realizing the propositional formula $P \rightarrow (Q \land R_i \land F_i)$. This can be done in \textsc{NEXPTIME}, by having a non-deterministic Turing machine guessing the local strategies of all processes, and verifying that such strategies satisfy the formula under all the (exponentially many) possible inputs of the environment. We show that the problem is also \textsc{NEXPTIME-hard}, via a reduction from the Dependency Quantifier Boolean Formula (DQBF) validity problem introduced in [20] to study time bounded multiplayer alternating machines. A DQBF is a quantified Boolean formula with a succinct description of dependencies between the quantified variables. Every DQBF has an equivalent form in which all existentially quantified variables are substituted by existentially quantified Skolem functions defined over their dependencies, and appearing at the beginning of the formula (e.g. $\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) \varphi(x_1, x_2, y_1, y_2)$ is a DQBF stating that $y_1$ depends on $x_1$, and has a functional form $\exists \sigma_1 \exists \sigma_2 \forall x_1 \forall x_2 \varphi(x_1, x_2, \sigma_1(x_1), \sigma_2(x_2))$ with $\sigma_1, \sigma_2$ the Skolem functions).

**Lemma 7.** Given an architecture $A$ and a formula $\phi \in \text{LTL}_{AG}$, deciding whether $\phi$ is realizable in $A$ is \textsc{NEXPTIME-hard}.

**Proof:** Consider any DQBF formula $\psi : \forall x_1 \ldots \forall x_k \exists y_1(x_1) \ldots \exists y_n(x_n) \varphi(x_1, \ldots, x_k, y_1 \ldots y_n)$ with $k$ universally quantified variables $x_i$ and $n$ existentially quantified variables $y_i$. We assume w.l.o.g. that the dependencies of each $y_i$ are only on some universally quantified variables $x_i$ and $n$ existentially quantified variables $y_i$. We construct the architecture $A = (P, p_i, E, V)$, where $P$ contains $n + 1$ processes, $V = \{x_i \in \psi\} \cup \{y_i \in \psi\}$, process $p_i$ receives as inputs from the environment all $x_i$, outputs variable $y_i$, while the environment uses all remaining $x_j$ as hidden variables. We construct the specification $\phi = \bigcup \varphi \in \text{LTL}_{AG}$. Both $A$ and $\phi$ are polynomial in the size of $\psi$. Because of Lemma 4,
\( \phi \) is realizable in \( A \) iff \( \varphi \) is realizable in \( A \). In turn, \( \varphi \) is realizable iff \( \psi \) is valid, with local strategies \( \sigma_i \) corresponding to the Skolem functions in the functional form of \( \psi \), and universal variables corresponding to all possible choices of the environment in \( A \). Since DQBF validity is NEXPTIME-hard [20], the statement follows.

Hence, we have the following result.

**Theorem 4.** Given an architecture \( A \) and a specification \( \phi \in LTL_{AG} \), the realizability of \( \phi \) in \( A \) is NEXPTIME-complete.

Observe that Lemma 6 reduces the problem of realizability of some \( \varphi \in LTL_{AG} \) to realizing a set of formulas of the form \( Q \), where \( Q \) is propositional. This in turn is reducible to DQBF validity (because of Lemma 4), and because of Lemma 7, the two problems are equivalent. In consequence, efficient algorithms for solving DQBF, such as [21], yield efficient synthesis procedures for LTL_{AG}, and vice versa. Moreover, if the DQBF tool outputs the corresponding Skolem functions, then a witness collective strategy for realizability can be obtained.

**VI. Conclusions**

In this paper we studied the distributed synthesis problem for relevant fragments of LTL. We presented a much finer characterization of undecidability results for distributed synthesis in terms of LTL fragments that uses eventually, globally and next operators. In contrast to previous decidability results that were non-elementary, we identify fragments where the complexity is EXPSPACE (or NEXPTIME-complete). An interesting direction of future work would be to develop algorithms for the problems for which we establish decidability, obtain efficient implementations of the algorithms for distributed synthesis problems, and finally consider some case-studies of practical examples.

**Acknowledgements.** The research was supported by Austrian Science Fund (FWF) Grant No P 23499- N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), Microsoft faculty fellows award, the Austrian Science Fund (FWF) Grant No P 23499- N23, FWF NFN (Rigorous Systems Engineering), the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

**References**


