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 modules [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 have been 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 the 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 EXPSAPCE) 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. Our result improves on previous decidability results (tower of exponentials), our proof technique is different, and our result is not obtained by reducing the problem to temporal logic. Moreover, our proof technique is not based on the classical proof technique of simulation.

II. MODEL DESCRIPTION

Architectures. An architecture \( A = (P, p_0, V, E) \) is a tuple where \( P = \{p_0, p_1, p_2, \ldots, p_n\} \) is a set of \( n+1 \) processes, \( p_0 \) is a distinguished process representing the environment. \( V \) is a set of (output) binary variables. \( E : P \times P \rightarrow 2^V \) 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 \( p \in P \), we denote with \( O(p) = \bigcup_{q \in P} E(p, q) \) the set of output variables of \( p \), and with \( I(p) = \bigcup_{q \in P} E(q, p) \) the set of input variables of \( p \). We require that for all \( p, q \in P \) : \( O(p) \cap O(q) = \emptyset \), i.e., no two processes write to the same variable. 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 \( \lambda(x) \). 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 \( p \in P^- \), a local strategy \( \sigma_p \) is a function \( \sigma_p : (2^{O(p)})^* \rightarrow 2^{V(p)} \), setting the output variables of \( p \) according to the history of its input variables. Observe that every such local strategy \( \sigma_p \) can be viewed as a labeling of an \( O(p) \)-labeled \( I(p) \)-tree \( T_{\sigma_p} \). A local strategy \( \sigma_p \) has finite memory if there exists a finite set \( M, m_0 \in M \), and functions \( f : M \times 2^{I(p)} \rightarrow M \) and \( g : M \rightarrow 2^{O(p)} \) such that for all \( x = x_1 x_2 \ldots x_k \) with \( x_i \in 2^{I(p)} \), 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. The collective strategy of the architecture \( A \) is a function \( \sigma : (2^{O(p)})^* \rightarrow 2^{V(O(p))} \), mapping every finite sequence of the outputs of the environment to a subset of the outputs of the processes \( p \) according to the composition \( \sigma_p : p \in P^- \). The collective strategy \( \sigma \) can be viewed as a \( (V \setminus O(p_0)) \)-labeled \( O(p_0) \)-tree \( T_\sigma \) 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 \( \sigma_p : p \in P^- \).

Synthesis (realizability). We will consider distributed reactive systems with specifications given by temporal logic formulae. For temporal logic formulae 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 \( (V \setminus O(p_0)) \)-labeled \( O(p_0) \)-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 \( p \in P^- \), then it is called realizable, and the collective strategy \( \sigma \) gives an implementation for \( \phi \) on \( A \).

III. SYNTHESIS FOR REACHABILITY SPECIFICATIONS

In the current section we discuss the synthesis problem for reachability specifications, where the objective consists of propositional formulae 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\(_\phi\). We consider LTL\(_\phi\) that consists of formulae \( \phi \) from the following LTL fragment:

\[
\begin{align*}
\theta & = P | \mathcal{X} P \\
\psi & = \theta_1 \land \theta_2 | \theta_1 \lor \theta_2 | -\theta \\
\phi & = Q \rightarrow \Diamond \psi
\end{align*}
\]

where \( P, Q \) are propositional formulae, \( \mathcal{X} \) 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 \# symbols. 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 $\#va\sqcupuv\#$, 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 $\pi_\perp$ over words $w$ from some alphabet containing $\sqcup$, such that $\pi_\perp(w)$ is the result of omitting all $\sqcup$ symbols from $w$. We define a scattered configuration $C$ of $M$ as a word over $\Sigma = \{0, 1, \sqcup, \sqcap, \sqcup\} \cup Q$ such that $\pi_\perp(C)$ 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 $\phi \in$ 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 $\sqcup$.

**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 $p_2$ concur 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 $\sqcup$ symbol.

**Conversion to LTL$_0$.** Next, we will provide the specification for the synthesis problem $\phi \in$ 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-url)
that the first two configurations of \( M \) have at most 9 letters 
\[ \#q_0 \vdash \#q_1 \wedge a \vdash \# \], 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, \( A_{safe} \) accepts unconditionally). Because of \( \mathcal{L} \), \( A_{safe} \) can verify that \( p_1 \) and \( p_2 \) conform with \( Cond_3 \) by checking the first 18 output letters. For the condition \( Cond_3 \), \( A_{safe} \) 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 \( A_3 \), verifies that \( Cond_3 \) holds at the current position, as follows. It ignores \( \perp \) symbols and compares whether \( C_1 \mid C'_1 \), configurations \( \pi_\perp(C_1) \) and \( \pi_\perp(C'_1) \) 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, \( C'_1 \not\rightarrow C_1 \), therefore \( A_3 \) accepts the word regardless of what follows. Otherwise, if those conditions hold, i.e., \( C'_1 \vdash C_1 \), \( A_3 \) non-deterministically verifies one of the following conditions: \( C'_2 \not\rightarrow C_2 \) or \( C_2 \not\rightarrow C'_2 \). Both conditions can be verified by safety automata, since \( C_2 \) and \( C'_2 \) either start concurrently, or \( C_2 \) is delayed by 1 step from \( C'_2 \). For the condition \( Cond_4 \) observe that if \( D \parallel D' \) and \( \pi_\perp(D) = \pi_\perp(D') \), then \( |D| - |D'| \leq 1 \) and the automaton needs to remember at most one symbol to compare \( \pi_\perp(D) \) and \( \pi_\perp(D') \). 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 \( p_1 \) and \( p_2 \) output sequences of scattered configurations whose \( \pi_\perp \) 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 \( p_1 \) and \( p_2 \) satisfy conditions \( Cond_0 \sim Cond_4 \). The lemma clearly holds for \( k = 1, 2 \), due to conditions \( Cond_0 \sim Cond_2 \). For the inductive step, assume that the lemma holds for \( k \geq 2 \). Consider a sequence of inputs to \( p_1 \) consisting of \( next \) signals only. Then, there is a sequence of inputs to \( p_2 \) consisting of some number of \( next \) signals and exactly \( \pi_\perp(C_k) \) \( \perp \) stall signals placed in a such way that \( p_1 \) outputs \( C_1 \ldots C_k C_{k+1} \), \( p_2 \) outputs \( C'_1 \ldots C'_{k-1} C'_k \), and \( C_k C_{k+1}, C'_{k-1} C'_k \) are synchronized, i.e. they start at the same position and \( C_k \parallel C_k' \parallel C_{k+1} \parallel C'_k \). By the induction assumption \( \pi_\perp(C'_k) \) and \( \pi_\perp(C_k) = \pi_\perp(C'_k) \) are, respectively, \((k-1)\)-th and \( k \)-th configurations of \( M \). Therefore, \( C_{k-1} \vdash C_k \) and, by \( Cond_3 \), \( C'_k \vdash C_{k+1} \). This implies that \( C_k C_{k+1} \) is the finite scattered preconfiguration and \( \pi_\perp(C_{k+1}) \) is the \((k+1)\)-th configuration of \( M \).

Given that for an input consisting of \( next \) signals only, \( p_1 \) outputs \( C_1 \ldots C_k C_{k+1} \) satisfying the statement, we can show that regardless of the number of \( stall \) signals, under condition \( \mathcal{L} \), \( p_1, p_2 \) output \( k + 1 \) scattered configurations satisfying the statement. First, the condition \( Cond_4 \) implies that if \( p_2 \) also has an input sequence consisting of \( next \) signals alone, it will output the same sequence, that is, \( C_1 \ldots C_k C_{k+1} \). By a simple induction on the number of \( stall \) signals each process receives, and condition \( Cond_4 \), we conclude that for any number of \( stall \) signals, as long as \( \mathcal{L} \) is satisfied by the environment, \( p_1, p_2 \) output \( k + 1 \) scattered configurations whose projections are the first \( k + 1 \) consecutive configurations of \( M \).

**Conversion to LTL\(_0\).** Given the safety automaton \( A_{safe} \) which verifies that \( \varphi \) is satisfied, we can construct a specification \( \phi \in LTL\(_0\) \) such that \( \phi \) 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 \( p_1, p_2 \)) variables \( q_1, \ldots, q_m \in E(p_e, p_e) \) to simulate the automaton \( A_{safe} \). We provide a high level description of the following formulæ:

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

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

\[ \psi_2 \text{ specifies that the current state of } A_{safe} \text{ is not rejecting, and } p_1 \text{ or } p_2 \text{ outputs a halting state of } M \text{ (i.e., some process reached a halting configuration of } M \text{, and both processes behaved according to } A_{safe}; \psi_2 \text{ is propositional}) \]

Finally, we construct \( \phi = Q \rightarrow \Diamond (\neg \psi_1 \lor \psi_2) \), with \( \phi \in LTL\(_0\) \). If \( \phi \) is realizable, the processes satisfy \( \psi_2 \) in all runs where the environment faithfully simulates \( A_{safe} \) and conforms with condition \( \mathcal{L} \) (i.e., \( Q \) and \( \psi_1 \) are true). Then \( p_1, p_2 \) 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 \( Cond_0 \). Hence, we obtain the following theorem.

**Theorem 1.** The realizability of specifications from \( LTL\(_0\) \) in \( A_0 \) 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 \( A \), the realizability of specifications from \( LTL\(_0\) \) in \( A \) is decidable iff \( A \) 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 formulae connected with Boolean operators, and the \( \chi \) temporal operators. 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 LTL\(\sqcap\). Then we show that the problem is decidable for a family of star architectures, despite the existence of information forks.

**Fragment LTL\(\sqcap\).** We consider LTL\(\sqcap\) that consists of formulae \(\phi\) from the following LTL fragment:
\[
\psi = P \mid \psi_1 \land \psi_2 \mid \psi_1 \lor \psi_2 \mid \neg \psi \mid X\psi
\]
\[\phi = Q \land \Box \psi\]
where \(P, Q\) are propositional formulae, 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 \(Q\) as the initial conditions. The fragment LTL\(\Box\) can express safety specifications, one of the most basic specifications 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., \(p_c\) is the central process, and \(\bigcup_i I(p_i) = O(p_c)\)) (Figure 2) and show that: (i) the realizability of some \(\phi \in\) LTL\(\Box\) 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.

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

**A. Overlapping inputs**

Here we demonstrate undecidability of realizability of specifications \(\phi \in\) LTL\(\Box\) for star architectures with overlapping inputs, and with \(\phi\) having \(X\)’-depth 1 (i.e., \(\phi\) belongs to a subclass of LTL\(\Box\) 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.

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

Given a Turing machine \(M\), recall the specification \(\varphi\) (from Section 3 for LTL\(\sqcap\)) encoding conditions \(\mathcal{L}\) and \(Cond_0 - Cond_4\) through the safety automaton \(A_{safe}\). In contrast with the previous section, here we require that process \(p_3\) (instead of \(p_c\)) faithfully simulates the safety automaton \(A_{safe}\) using the output variables \(q_1, \ldots, q_m \in E(p_3, p_c)\). Note that \(A_{safe}\) 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).

**Formal proof.** We provide a high level description of the following formulae:
\[
Q\text{ specifies that the first state of } A_{safe}\text{ according to the output variables } \{q_1, \ldots, q_m\} \text{ is compatible with the initial values of } x_1, x_2, y_1 \text{ and } y_2 \text{ (i.e., } \{q_1, \ldots, q_m\}\text{ represent the state of } A_{safe}\text{ reached from the initial state after reading the initial values of } x_1, x_2, y_1 \text{ and } y_2; Q \text{ is propositional)}
\]
\[
\psi_1 \text{ specifies that } A_{safe}\text{ has a transition from the current state to the next state, encoded by the values of } \{q_1, \ldots, q_m\}\text{ in the current and the next round, according to the value of variables } x_1, x_2, y_1 \text{ and } y_2 \text{ in the next round (i.e., } p_c\text{ simulates } A_{safe}\text{ faithfully); } \psi_1\text{ contains only propositional and non-nested } \mathcal{X} \text{ operators).
\]
\[
\psi_2 \text{ specifies that } p_1 \text{ and } p_2 \text{ do not output a halting state of } M\text{ (i.e., } M \text{ does not terminate; } \psi_2 \text{ is propositional)}
\]
\[
\psi_3 \text{ specifies that } A_{safe}\text{ does not reach a rejecting state (i.e., the processes conform to conditions } Cond_0-Cond_4\text{ or the environment violates } \mathcal{L}; \psi_3 \text{ is propositional)}
\]

We construct \(\phi = Q \land \Box (\psi_1 \land \psi_2 \land \psi_3)\). Similarly as in the case of LTL\(\sqcap\), if \(\phi\) is realizable, \(p_3\) faithfully simulates \(A_{safe}\) (\(Q\) and \(\psi_1\) are true), and \(p_1, p_2\) satisfy \(\varphi\) in all runs where the environment conforms with condition \(\mathcal{L}\) (\(\psi_3\) is true). By Lemma 1, \(p_1\) and \(p_2\) 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, \(\phi\) is realizable by local strategies where (i) \(p_1, p_2\) output a legal sequence of configurations of \(M\) and conform with condition \(Cond_0\), and (ii) \(p_3\) faithfully simulates \(A_{safe}\). Hence we have the following result.

**Theorem 2.** The realizability of specifications from LTL\(\Box\) 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, the realizability problem for LTL\(\Box\) in \(A_1\) remains undecidable even if we restrict the strategies to be finite-memory. We refer to the longer version of this paper in [19] for the proof.

**Information fork-meet.** We say that an architecture \(A = (P, p_c, \mathcal{V}, \mathcal{E})\) has an information fork-meet if there are three processes \(p_1, p_2, p_3 \in \mathcal{P}^-\) 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 \(p_c\),
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 LTL₀, 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 \( p₁, p₂ \) and \( p₃ \) accordingly.

**Corollary 2.** The realizability of LTL\( _{\Box} \) specifications in architectures containing an information fork-meet is undecidable.

**B. Pairwise disjoint inputs**

In this subsection we discuss synthesis for formulae \( \phi \in \text{LTL}_{\Box} \) 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 \( S_n \). Our goal is to prove decidability of realizability of such \( \phi \in \text{LTL}_{\Box} \) in every architecture \( A \in S_n \), by showing that whenever such \( \phi \) is realizable, it admits strategies of bounded memory.

Consider some architecture \( A \in S_n \) and an arbitrary \( \phi = Q \land \Box \psi \in \text{LTL}_{\Box} \), with the nesting level of \( \land \) 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 \( \text{O}(p_i)\)-labeled \( \text{I}(p_i)\)-trees \( T_{\sigma_i} \). We will show how to construct strategies \( \tau_i \) that also realize \( \phi \), where each tree \( \text{I}(p_i)\)-tree \( T_{\tau_i} \) representing \( \tau_i \) is defined from first \( 2^{|\mathcal{P}|}\cdot|\mathcal{L}| + 1 \) levels of \( T_{\sigma_i} \), by applying a unfolding function given below. We first define the notion of a computation that is not the empty set in some computation.

**Definition 1.** For a computation \( \ell(\pi) \) and some \( i \in \mathcal{N} \) we say that \( i \) closes \( \neg \psi \) in \( \ell(\pi) \) if \( \ell(\pi)[i \to k, \infty] = \neg \psi \).

**Remark 2.** \( \ell(\pi) \models \Box \psi \iff \text{no } i \text{ closes } \neg \psi \text{ in } \ell(\pi) \).

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 \( \text{O}(p_i)\)-labeled \( \text{I}(p_i)\)-tree \( T_{\sigma_i} \). For every node \( x \in T_{\sigma_i} \), with \( |x| \geq k \), we denote with \( \mathbf{\tau}(x) = \{x_k, x_{k-1}, \ldots, x_1\} \) the \( k \)-node suffix of the unique path to \( x = x_1 \), and define the type of \( x \) under \( \sigma \) as \( \ell_{\sigma_i}(x) = \mathbf{\tau}(x) \).

For every level \( l \geq k \), we define the type of \( l \) under \( \sigma \) as \( \ell_{\sigma}(l) = \{\ell_{\sigma_i}(x) \mid i \in \{1, \ldots, n\}, x \in T_{\sigma_i}, \text{ and } |x| = l\} \), i.e., the type of a level \( l \) is the set of the types of the nodes of level \( l \) of every \( T_{\sigma_i} \), where \( i \in \{1, \ldots, n\} \). Note that there exist at most \( 2^{|\mathcal{L}|} \cdot 2^{|\mathcal{P}|} \) distinct types of nodes. Consequently, there exist at most \( 2^{|\mathcal{L}|} \cdot 2^{|\mathcal{P}|} \) distinct types of levels.

We naturally extend the definition of types to nodes of the \( (V \setminus \text{O}(p_e))\)-labeled \( \text{O}(p_e)\)-tree \( T_{\sigma} \) as \( \ell_{\sigma}(\pi) = \ell_{\sigma}(\pi[e]) \).

Consider some computation \( \ell_{\sigma}(\pi) \in T_{\sigma} \). Observe that whether some \( i \) closes \( \neg \psi \) in \( \pi \) depends only on the \( \ell_{\sigma_i}(\pi[e]) \) i.e., the type \( \ell_{\sigma_i}(\pi[e]) \) determines whether \( i \) closes \( \neg \psi \) in \( \pi \). Hence, we have the following remark:

**Remark 3.** For a formula \( \phi \in \text{LTL}_{\Box} \) there exists a set of types \( \Delta \) such that for every tree \( T_{\sigma} \), a path \( \pi \) in \( T_{\sigma} \) satisfies \( \phi \) if \( \ell_{\sigma}(\pi[1]) \models Q \) and for all \( i \in \mathcal{N} \), we have \( \ell_{\sigma}(\pi[e]) \in \Delta \), i.e., the set of types of nodes in \( T_{\sigma} \) is a subset of \( \Delta \).

**Folding function.** Assume that there exist two levels \( l_1 < l_2 \) such that \( \ell_{\sigma}(l_1) = \ell_{\sigma}(l_2) \). Then for every tree \( T_{\sigma} \), for every node \( x \) in level \( l_2 \), there exists a node \( y \) in level \( l_1 \) such that \( \ell_{\sigma_i}(x) = \ell_{\sigma_i}(y) \), 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 \( f_i : (2^{|\mathcal{P}|})^* \rightarrow (2^{|\mathcal{P}|})^* \) recursively as follows:

\[
f_i(x) = \begin{cases} x & \text{if } |x| < l_2 \\ y & \text{if } |x| = l_2 \text{ where } |y| = l_1 \text{ and } \ell_{\sigma_i}(x) = \ell_{\sigma_i}(y) \\ f_i(f_i(y)z) & \text{if } |x| > l_2 \text{ for } x = yz \text{ with } z \in 2^{|\mathcal{P}|} \end{cases}
\]

and construct local strategies \( \tau_i(x) = \ell_{\sigma_i}(f_i(x)) \). 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. Since the collective strategies \( \sigma \) and \( \tau \) behave identically on the first \( l_1 \) levels, \( \tau \) realizes the propositional \( Q \).

The following analysis focuses on the \( \Box \psi \) part of \( \phi \).

The strategies \( \tau_i \) preserve the types under \( \sigma_i \) 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 \( \sigma \) as well. 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 \( \phi \), as it does not introduce new types. We formalize these arguments below.

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

**Lemma 2.** For every \( x \in (2^{|\mathcal{P}|})^* \) with \( |x| \geq k \), we have that \( \tau_i(x) = f_i(\ell_{\sigma}(f_i(x))) \).

**Proof:** Our proof proceeds by induction on \( |x| \):

1. \( |x| < l_2 \): For all nodes \( w \in \mathbf{\tau}(x) \), we have that \( \tau_i(w) = \sigma_i(f_i(w)) = \sigma_i(\ell_{\sigma}(\mathbf{\tau}(x))) \), hence \( \ell_{\sigma_i}(\mathbf{\tau}(w)) = \ell_{\sigma_i}(\mathbf{\tau}(x)) \) and thus \( \tau_i(x) = f_i(\ell_{\sigma}(\mathbf{\tau}(x))) \).

2. \( |x| = l_2 \): The statement holds by definition.

3. \( |x| = m + 1 \): Let \( x = yz \) with \( |y| = m \). By the inductive hypothesis, \( \tau_i(y) = f_i(\ell_{\sigma}(f_i(x))) \).

By construction, \( \tau_i(x) = f_i(\ell_{\sigma}(f_i(x))) \) (i.e., the strategy \( \tau_i \) will label \( x \) as \( \sigma_i \) labels its image \( f_i(x) \), and the types of these two nodes are equal).

We denote the result follows.

The following remark observes that for every architecture from \( S_n \), every node in the collective strategy tree corresponds to a unique set of nodes in the local strategy trees and vice versa, and that the collective strategy on that node equals the union of the local strategies on the corresponding local nodes.

**Remark 4.** The following assertions hold:

1. For every global node \( x = x_1^2 \cdot x_2^3 \cdot \ldots \cdot x_m^m \) in \( T_{\sigma} \), for every \( x_i^j \in (2^{|\mathcal{P}|})^* \), for every tree \( T_{\tau_i} \), there exists a (unique) node \( x_j = x_1^j \cdot x_2^j \cdot \ldots \cdot x_m^j \) such that \( x_j = x^j \cap 2^{|\mathcal{P}|} \), and
Moreover, for every collective strategy \( \sigma \), we have \( \sigma(x) = \bigcup_j \sigma_j(x_j) \).

It follows from the above remark and Lemma 2, that for every \( x \in T_\sigma \), we have that \( t_\sigma(x) = t_j(f(x)) \), where \( f(x) = \bigcup_j f_j(x_j) \). That is, the local folding functions \( f_j \) 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 \( \phi \in \text{LTL} \) exist levels \( l_1 \) and \( l_2 \) with the same type under \( \sigma \), we can construct a collective strategy \( \tau \) for which every local strategy \( \tau_i \) uses only the first \( l_2 \) 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 \( 2^{2|V|+1} \), and thus every local strategy \( \tau_i \) operates in the first \( 2^{2|V|+1} \) 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, \( 2^{k|V|} \). This leads to Theorem 3 (we refer to [19] for the formal proof).

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

V. SYNTHESIS WITHOUT THE NEXT OPERATOR

In the current section we consider a fragment of 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\(_{\neg X}\).** We consider LTL\(_{\neg X}\) that consists of formulas \( \phi \) from the following LTL fragment:

\[
\phi = \bigwedge_i \Box P_i \rightarrow \left( \bigwedge_i \Box Q_i \land \Box R_i \land \bigwedge_i \Box F_i \right) \\
\equiv \bigwedge_i P_i \rightarrow \left( \bigwedge_i Q_i \land \bigwedge_i R_i \land \bigwedge_i F_i \right) \\
\equiv \Box P \rightarrow \left( \Box Q \land \Box R \land \Box F \right)
\]

for \( i \in \{1, \ldots, m\} \), with \( P_i, Q_i, R_i, F_i \) propositional formulae, and \( P = \bigwedge_i P_i, Q = \bigwedge_i Q_i \). We consider the standard semantics of LTL. The LTL\(_{\neg X}\) can express specifications that consist of conjunction of safety assumptions, and guarantees where each guarantee is a safety, reachability, or a liveness condition.

A propositional formula \( Q \) has the property that can either be realized in a single step, or is not realizable. This implies that realizable formulae \( \Box Q \) admit memoryless strategies which repeat the single step realization of \( Q \). A similar argument establishes that reachability and safety specifications of propositional formulae are equivalent with respect to realizability. We formally state these observations in Lemmas 4 and 5, and refer to [19] for the proofs.

**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.

**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 \( A \).

Lemma 6 shows that the realizability of some \( \phi \in \text{LTL}_{\neg X} \) reduces to realizing a set of safety formulae of the form of Lemma 4.

**Lemma 6.** Let \( A \) be any architecture and \( \phi = \Box P \rightarrow (\Box Q \land \bigwedge_i \Box R_i \land \bigwedge_i \Box F_i) \in \text{LTL}_{\neg X} \). The formula \( \phi \) 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 \( \sigma_{R_i}^R \) and \( \sigma_{F_i}^F \) for every process \( p_j \), such that the collective strategy \( \sigma_{R_i}^R \) implements \( \phi_{R_i} \), and the collective strategy \( \sigma_{F_i}^F \) implements \( \phi_{F_i} \). Construct local strategies \( \tau_j \) such that for every \( x = yz \) with \( |z| = (1 + |x|) \mod 2m \), we have \( \tau_j(x) = \sigma_{R_i}^{R_i}(z) \) if \( |z| \leq m \), and \( \tau_j(x) = \sigma_{F_i}^{F_i}(z) \) if \( |z| > m \). i.e. the local strategy \( \tau_j \) repeatedly alternates between all the strategies \( \sigma_{R_i}^{R_i} \) in the first \( m \) steps, and between all the strategies \( \sigma_{F_i}^{F_i} \) the next \( m \) steps. Let \( \tau \) be the collective strategy of all \( \tau_j \) and consider an arbitrary path \( \pi \) in \( T \). Either \( \ell_x(\pi)[k] = \neg P \) for some \( k \), or for all \( k \), it holds \( \ell_x(\pi)[k] = P \), and by construction, for \( i = 1 + k \mod 2m \), we have \( \ell_x(\pi)[k] = Q \land R_i \) when \( i \leq m \) and \( \ell_x(\pi)[k] = Q \land F_{i-m} \) when \( i > m \). In both cases, \( \ell_x(\pi) = \phi \).

(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 \( T_\sigma \), such that for all
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 \( \phi_F, \phi_R_i \), by providing a constructive argument. As a consequence of Lemma 4, deciding whether every \( \phi_F, \phi_R_i \) is realizable reduces to realizing the propositional formulae \( (P \to (Q \land R_i)) \) and \( (P \to (Q \land F)) \). This can be done in \text{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 \text{NEXPTIME}-hard, via a reduction from the Dependency Quantifier Boolean Formula (DQBF) validity problem introduced in [20] to study time bounded multi-player 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 moved to the beginning of the formula (e.g., \( \forall x_1 \forall x_2 \exists y_1 (x_1) \exists y_2 (x_2) \phi(x_1, x_2, y_1, y_2) \) is a DQBF stating that \( y_i \) depends on \( x_i \), and has a functional form \( \exists \sigma_1 \exists \sigma_2 \forall x_1 \forall x_2 \phi(x_1, x_2, \sigma_1(x_1), \sigma_2(x_2)) \) 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 \text{NEXPTIME}-hard.

**Proof:** Consider any DQBF formula \( \psi = \forall x_1 \ldots \forall x_k \exists y_1 (x_1) \ldots \exists y_n (x_n) \phi(x_1, \ldots, x_k, y_1, \ldots, y_n) \) 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 \( x_i \). We construct the architecture \( A = \langle P, p_i, V, E \rangle \), 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 \( \phi = \square \psi \in \text{LTL}_{AG} \). Both \( A \) and \( \phi \) are polynomial in the size of \( \psi \). Because of Lemma 4, \( \phi \) is realizable in \( A \) iff \( \psi \) is realizable in \( A \). In turn, \( \phi \) 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 \text{NEXPTIME}-hard [20], the statement follows.

Hence, we have the following result.

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

Observe that Lemma 6 reduces the problem of realizability of some \( \psi \in \text{LTL}_{AG} \) to realizing a set of formulae of the form \( \square Q \), 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 \text{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 \text{LTL}. We presented a much finer characterization of undecidability results for distributed synthesis in terms of \text{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 \text{EXPSPACE} (or \text{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.

**Acknowledgments.** 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 NFN RiSE (Rigorous Systems Engineering), the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

**References**