IST AUSTRIA

Institute of Science and Technology

Distributed Sytnehsis for LTL Fragments

Krishnendu Chatterjee and Thomas A. Henzinger and Jan Otop and Andreas Pavlogiannis

Technical Report No. IST-2013-130-v1+1
Deposited at 08 Jul 2013 08:35
http://repository.ist.ac.at/130/1/Distributed$_{synthesis.pdf}$

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Copyright © 2012, by the author(s).
All rights reserved.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

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 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 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 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* is a tuple $\mathcal{A} = (\mathcal{P}, p_e, V, E)$, where $\mathcal{P} = \{p_e, p_1, p_2, \dots p_n\}$ 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 $E : \mathcal{P} \times \mathcal{P} \to 2^V$ 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 $p \to q$ 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 $x \in \left(2^B\right)^*$. A (possibly infinite) sequence of nodes $\pi = (x_1, x_2 \dots)$ 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 π, we will use $\pi[i]$ to denote the element of π in the i-th position, while $\pi[i, \infty]$ denotes the infinite suffix of π starting at position i. An *A-labeled B-tree* T_λ is a B-tree equipped with a labeling function of its nodes, $\lambda : \ \left(2^B\right)^* \to 2^A$. 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 ℓ_λ of x consists of the branch z from the parent and the label λ of x. For a (possibly infinite) path $\pi = (x_1, x_2, \dots)$, we define with $\ell_\lambda(\pi) = (\ell_\lambda(x_1), \ell_\lambda(x_2) \dots)$.

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

Collective strategies. Every such local strategy σ_p can be viewed as a labeling of an $O(p)$-labeled $I(p)$-tree T_{σ_p}. The *collective strategy* of the architecture $\mathcal{A}$ is a function $\sigma : \left(2^{O(p_e)}\right)^* \to 2^{V \setminus O(p_e)}$, 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 σ can be viewed as a $(V \setminus O(p_e))$-labeled $O(p_e)$-tree T_σ and for any infinite path π in T_σ, we will call $\ell_\sigma(\pi)$ a *computation*. Hence, T_σ describes a distributed algorithm, and every infinite path $\pi = (x_1, x_2, \dots)$ 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 ϕ in an architecture $\mathcal{A}$ asks whether there exist local strategies σ_p for every process p, such that for every infinite path π of the $(V \setminus O(p_e))$-labeled $O(p_e)$-tree T_σ of the collective strategy σ, with π starting from the root, we have $\ell_\sigma(\pi) \models \phi$. If ϕ admits such strategies σ_p for every $p \in \mathcal{P}^-$, then it is called *realizable*, and the collective strategy σ gives an *implementation* for ϕ on $\mathcal{A}$.

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$_\Diamond$. We consider LTL$_\Diamond$ that consists of formulas ϕ from the following LTL fragment:

$$\begin{aligned} \theta &= P \mid \mathcal{X} P \\ \psi &= \theta_1 \wedge \theta_2 \mid \theta_1 \vee \theta_2 \mid \neg \theta \\ \phi &= Q \to \Diamond \psi \end{aligned}$$

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 $\mathcal{Q}$ 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 $\#v\mathbf{q}au\sqcup\#$, where $a \in \{0,1\}$, $v,u \in \{0,1\}^*$ and $\mathbf{q} \in \mathcal{Q}$. Such a configuration has the standard interpretation as an infinite tape such that v is the part of the tape preceding the head, $\mathbf{q}$ 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 $\perp$, such that $\pi_\perp(w)$ is the result of omitting all $\perp$ symbols from w. We define a *scattered configuration* C of M as a word over $\Sigma = \{0,1,\sqcup,\perp,\#\} \cup \mathcal{Q}$ 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$_\Diamond$. This is obtained through a reduction from the halting problem of M, by constructing a specification $\phi \in$ LTL$_\Diamond$ 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 $\{0,1,\sqcup,\#\} \cup \mathcal{Q}$ of the current configuration of M, or it should output $\perp$.

Construction of φ. First, we will provide a regular safety property φ 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 φ does not belong to the LTL$_\Diamond$ fragment, but we will show how it can be expressed by a safety automaton A_{safe}. Then, we will prove that if φ is realizable, and the environment conforms with the alternation assumption, then the processes output a legal sequence of configurations of M, scattered with the $\perp$ symbol.

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

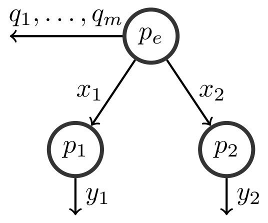


Fig. 1: The architecture A_0 which consists an information fork.

Formal proof. A safety automaton cannot express a scattered configuration that is finite. Thus, we define a *scattered preconfiguration 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 Σ that begins with $\#$, there is at most one symbol from $\mathcal{Q}$, 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 $\perp(C)$ the set of positions in C where $\perp$ occurs, and write $C_1 \parallel C_2$ if the symmetric difference of $\perp(C_1)$ and $\perp(C_2)$ has at most one element, i.e., $|\perp(C_1) \triangle \perp(C_2)| \leq 1$. We define as $C_1 \vdash C_2$ if $C_1 \parallel C_2$ and

(i) $\pi_\perp(C_2)$ follows legally from $\pi_\perp(C_1)$ 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 $\pi_\perp(C_2')$ follows legally from $\pi_\perp(C_1')$.

For infinite words w_1, w_2, we define $w_1 \otimes w_2$ as a word over $\Sigma \times \Sigma$ such that the i-th letter of $w_1 \otimes w_2$ 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 $\{C_1 \otimes C_2 : C_1 \parallel C_2\}$ and $\{C_1 \otimes C_2 : C_1 \vdash C_2\}$.

Construction of φ. We first construct the regular safety property $\varphi = \mathcal{L} \to \bigwedge_{0 \leq i \leq 4} Cond_i$, where $\mathcal{L}$ (the alternation assumption) and $Cond_i$ are defined as follows:

$\mathcal{L}$: for every process, every *stall* signal is followed by a *next* signal.
$Cond_0$: each process outputs $\perp$ when its input is *stall*, otherwise it outputs a letter from $\Sigma \setminus \{\perp\}$,
$Cond_1$: each process produces a sequence of scattered preconfigurations,
$Cond_2$: initially, each process produces two scattered configurations of M, whose projections are the first two valid configurations of M,
$Cond_3$: if starting from some position i, p_1 outputs consecutively C_1, C_2 and p_2 outputs consecutively C_1', C_2', then $C_1' \vdash C_1$ implies $C_2' \vdash C_2$ or $C_2' \not\parallel C_2$,
$Cond_4$: if D, D' are outputs of p_1, p_2 up to some positions such that $D \parallel D'$ and $|\pi_\perp(D)| = |\pi_\perp(D')|$, then $\pi_\perp(D) = \pi_\perp(D')$.

We provide a high-level description of the construction of an alternating safety automaton A_{safe} (see [18] for the definition of alternating automata) which verifies that every execution satisfies φ. Note that A_{safe} can be transformed to a non-deterministic automaton by a standard power-set construction. Clearly, conditions $\mathcal{L}$, $Cond_0$ and $Cond_1$ can be expressed by a safety automaton. For the condition $Cond_2$, observe that the first two configurations of M have at most 9 letters $\#\mathbf{q_0} \sqcup \#\#\mathbf{q_1} a \sqcup \#$, 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 $Cond_2$ 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 $\mathcal{A}_3$, verifies that $Cond_3$ holds at the current position, as follows. It ignores $\perp$ symbols and compares whether $C_1 \parallel C'_1$, configurations $\pi_\perp(C_1)$ and $\pi_\perp(C'_1)$ are equal everywhere except for positions adjunct to the head of M, and the letters adjunct to the head are consistent with the transition of M. If one of these conditions is violated, $C'_1 \not\vdash C_1$, therefore $\mathcal{A}_3$ accepts the word regardless of what follows. Otherwise, if those conditions hold, i.e., $C'_1 \vdash C_1$, $\mathcal{A}_3$ non-deterministically verifies one of the following conditions: $C'_2 \not\parallel C_2$ or $C'_2 \vdash C_2$. 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 φ is realizable, then for every $k \in \mathcal{N}$, 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 $Cond_0$-$Cond_4$. The lemma clearly holds for $k = 1, 2$, due to conditions $Cond_0 - Cond_2$. 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 $|\pi_\perp(C_k)|$ *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 $C_k \parallel C'_{k-1}$, $C_{k+1} \parallel C'_k$. By the induction assumption $\pi_\perp(C'_{k-1})$ 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+1} is a 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$_\Diamond$. Given the safety automaton A_{safe} which verifies that φ is satisfied, we can construct a specification $\phi \in$ LTL$_\Diamond$, such that ϕ 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 $q_1, \ldots, q_k \in E(p_e, p_e)$ to simulate the automaton A_{safe}. We provide a high level description of the following formulas:

- Q specifies that the first state of A_{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_{safe} reached from the initial state after reading the initial values of x_1, x_2, y_1 and y_2; Q is propositional)
- ψ_1 specifies that A_{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_{safe} faithfully; ψ_1 contains only propositionals and non-nested $\mathcal{X}$ operators).
- ψ_2 specifies that the current state of A_{safe} is not rejecting, and p_1 or p_2 output a halting state of M (i.e., a halting configuration of M was reached by some process, and both processes behaved according to A_{safe}; ψ_2 is propositional).

Finally, we construct $\phi = Q \rightarrow \Diamond(\neg\psi_1 \vee \psi_2)$, with $\phi \in$ LTL$_\Diamond$. If ϕ is realizable, the processes satisfy ψ_2 in all runs where the environment faithfully simulates A_{safe} and conforms with condition $\mathcal{L}$(i.e., Q and ψ_1 are true). Then p_1, p_2 output a halting state of M and satisfy φ, 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 ϕ 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$_\Diamond$ 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 φ, 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 LTL$_\Diamond$ in $\mathcal{A}$ is decidable iff $\mathcal{A}$ 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 $\mathcal{X}$ 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 LTL$_\Diamond$. Then we show that the problem is decidable for the family of star architectures, despite the existence of information forks.

Fragment LTL$_\Box$. We consider LTL$_\Box$ that consists of formulas ϕ from the following LTL fragment:

$$\psi = P \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \neg\psi \mid \mathcal{X}\psi$$
$$\phi = Q \wedge \Box\psi$$

where P, Q are propositional formulas, and $\Box$ is the globally operator. We consider the standard semantics of LTL. The $\Box\psi$ part of ϕ specifies a safety condition, and we interpret the Q part as the initial conditions. The fragment LTL$_\Box$ 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 $\bigcup_i I(p_i) = O(p_e)$) (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*.

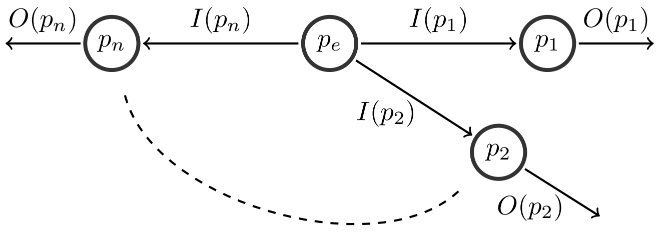


Fig. 2: The family of start architectures S_n.

A. Overlapping inputs

Here we demonstrate undecidability of realizability of specifications $\phi \in$ LTL$_\Box$ for star architectures with overlapping inputs, and with ϕ having $\mathcal{X}$-depth 1 (i.e., ϕ belongs to a subclass of LTL$_\Box$ where $\mathcal{X}$ 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 φ (from Section 3 for LTL$_\Diamond$) 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_e) faithfully simulates the safety automaton A_{safe} using the output variables $q_1, \dots q_m \in E(p_3, p_e)$. 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).

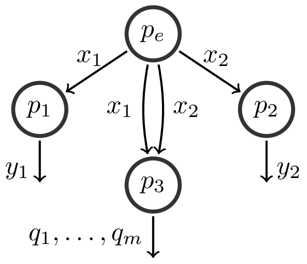


Fig. 3: The architecture A_1 consists an information fork-meet.

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

- Q specifies that the first state of A_{safe} according to the output variables $\{q_1, \dots q_m\}$ is compatible with the initial values of x_1, x_2, y_1 and y_2 (i.e. $\{q_1, \dots q_m\}$ represent the state of A_{safe} reached from the initial state after reading the initial values of x_1, x_2, y_1 and y_2; Q is propositional)
- ψ_1 specifies that A_{safe} has a transition from the current state to the next state, encoded by the values of $\{q_1, \dots q_m\}$ 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_{safe} faithfully; ψ_1 contains only propositionals and non-nested $\mathcal{X}$ operators).
- ψ_2 specifies that p_1 and p_2 do not output a halting state of M (i.e., M does not terminate; ψ_2 is propositional).
- ψ_3 specifies that A_{safe} does not reach a rejecting state (i.e., the processes conform to conditions $Cond_0$-$Cond_4$ or the environment violates $\mathcal{L}$; ψ_3 is propositional).

We construct $\phi = Q \wedge \Box(\psi_1 \wedge \psi_2 \wedge \psi_3)$. Similarly as in the case of LTL$_\Diamond$, if ϕ is realizable, p_3 faithfully simulates A_{safe} (Q and ψ_1 are true), and p_1, p_2 satisfy φ in all runs where the environment conforms with condition $\mathcal{L}$ (ψ_3 is true). By Lemma 1, p_1 and p_2 output a legal sequence of configurations of M, and ψ_2 guarantees that M does not halt. In the inverse direction, if M does not halt, then ϕ is realizable by local strategies for which (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, we show that even if we restrict the problem to finite-memory strategies, the realizability problem for LTL$_\Box$ 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 $\widehat{N}$ that simulates N and counts the number of steps taken by N. This counter is stored on the tape of $\widehat{N}$. If N halts, $\widehat{N}$ loops at the last configuration and does not increase the counter. Clearly, N started on the empty tape halts iff $\widehat{N}$ 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 σ_1, σ_2 that realize the specification ϕ iff 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 σ_1, σ_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 ϕ. Conversely, assume that σ_1, σ_2 are finite-memory strategies realizing the specification ϕ. 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 σ_1 and σ_2, or it is infinite because a strategy loops. As σ_1, σ_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 σ_1 and σ_2.

Information fork-meet. We say that an architecture $\mathcal{A} = (\mathcal{P}, p_e, V, E)$ has an *information fork-meet* if there are three processes $p_1, p_2, p_3 \in \mathcal{P} \setminus \{p_e\}$ and paths π_1, π_2 in the underlying graph such that

1) the first edges in π_1, π_2 are labeled by output variables of p_e,
2) the last edge of π_1 is an input variable of p_1, but not p_2
3) the last edge of π_2 is an input variable of p_2, but not p_1
4) the last edges of π_1, π_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$_\Diamond$, the undecidablity argument can be carried to any architecture containing such a structure, by introducing additional conditions in φ 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 $\overline{S}_n$. Our goal is to prove decidability of realizability of such $\phi \in$ LTL$_\Box$ in every architecture $\mathcal{A} \in \overline{S}_n$, by showing that whenever such ϕ is realizable, it admits strategies of bounded memory.

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

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

Remark 2. *$\ell(\pi) \models \phi$ iff no i closes $\neg\psi$ in $\ell(\pi)$.*

Let $\sigma_1, \ldots, \sigma_n$ be local strategies and σ be the collective strategy induced by $\sigma_1, \ldots, \sigma_n$. For every $i \in \{1, \ldots, n\}$, the local strategy σ_i is represented by an $O(p_i)$-labeled $I(p_i)$-tree T_{σ_i}. For every node $x \in T_{\sigma_i}$, with $|x| \geq k$, we denote with $\overline{\pi}_x = (x_k, x_{k-1} \ldots x_1)$ the k-node suffix of the unique path to $x = x_1$, and define the *type* of x under σ_i as $t_{\sigma_i}(x) = \ell_{\sigma_i}(\overline{\pi}_x)$. For every level $l \geq k$ we define the type of l under σ as $t_\sigma(l) = \{t_{\sigma_i}(x) : i \in \{1, \ldots, n\}, x \in T_{\sigma_i}$ 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_{σ_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 $2^{2^{k|V|}}$ distinct types of levels.

We naturally extend the definition of types to nodes of the $(V \setminus O(p_e))$-labeled $O(p_e)$-tree T_σ as $t_\sigma(x) = \ell_\sigma(\overline{\pi}_x)$. Consider some computation $\ell_\sigma(\pi)$ in T_σ. Observe that whether some i closes $\neg\psi$ in π depends only on the $\ell_\sigma(\pi)[i]$ i.e., the type $t_\sigma(\pi[i])$ determines whether i closes $\neg\psi$ in π. Hence, we have the following remark:

Remark 3. *For a formula $\phi \in$ LTL$_\Box$ there exists a set of types Δ such that for every tree T_σ, a path π in T_σ satisfies ϕ if for $i \in \mathcal{N}$, we have $t_\sigma(\pi[i]) \in \Delta$, i.e., the set of types of nodes in T_σ is a subset of Δ.*

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_{σ_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 $f_i : \left(2^{I(p_i)}\right)^* \to \left(2^{I(p_i)}\right)^*$ recursively as follows:

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

and construct local strategies $\tau_i(x) = \sigma_i(f_i(x))$. Hence, every strategy τ_i behaves as σ_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 τ_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 τ as well. Observe that the set of all such types serves as the set Δ of Remark 3, which in turn guarantees that the collective strategy τ also realizes ϕ. We formalize these arguments below.

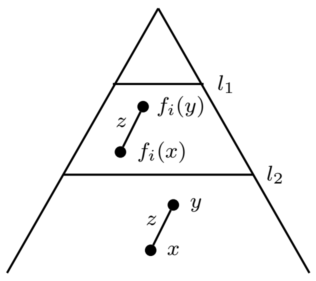
The following lemma establishes that for all nodes x in all T_{τ_i}, the type of x is the same as the type of its image under f_i in the corresponding T_{σ_i}.

Lemma 2. *For every $x \in \left(2^{I(p_i)}\right)^*$ with $|x| \geq k$, we have that $t_{\tau_i}(x) = t_{\sigma_i}(f_i(x))$.*

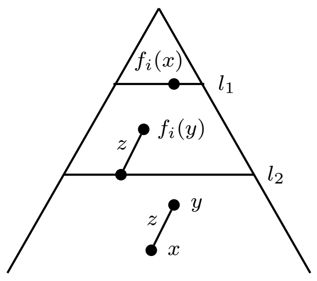
Proof: Our proof proceeds by induction on $|x|$:

1) $|x| < l_2$: For all nodes w in $\overline{\pi}_x$, we have that $\tau_i(w) = \sigma_i(f_i(w)) = \sigma_i(w)$, hence $\ell_{\tau_i}(\overline{\pi}_x) = \ell_{\sigma_i}(\overline{\pi}_x)$ and thus $t_{\tau_i}(x) = t_{\sigma_i}(f(x))$.
2) $|x| = l_2$: The statement holds by definition.
3) $|x| = m+1$: Let $x = yz$ with $|y| = m$. By the inductive hypothesis, $t_{\tau_i}(y) = t_{\sigma_i}(f_i(y))$. 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 $\tau_i(x) = \sigma_i(f_i(x)) = \sigma_i(f_i(y)z)$, thus $t_{\tau_i}(x) = t_{\sigma_i}(f_i(y)z) = t_{\sigma_i}(f_i(x))$ (i.e., the strategy τ_i will label x as σ_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, $t_{\sigma_i}(f_i(x)) = t_{\sigma_i}(f_i(y)z)$ (i.e., $f_i(y)$ extended by z hits level l_2, and the folding function f_i will bring x to level l_1, to a node of the same type). Then $\tau_i(x) = \sigma_i(f_i(x)) = \sigma_i(f_i(y)z)$, hence as in (i), $t_{\tau_i}(x) = t_{\sigma_i}(f_i(y)z) = t_{\sigma_i}(x)$.

The desired result follows. ∎



(a) Case (i)



(b) Case (ii)

Fig. 4: The two cases of the inductive step of Lemma 2.

The following remark observes that for every architecture from $\overline{S}_n$, 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 $x = x^1x^2\ldots x^m$ in T_σ with every $x^i \in 2^{O(p_e)}$, for every tree T_{σ_j}, there exists a (unique) node $x_j = x_j^1x_j^2\ldots x_j^m$ such that $x_j^i = x^i \cap 2^{I(p_j)}$, and*
2) *for every set of nodes $\{x_j = x_j^1x_j^2\ldots x_j^m\}$ with one x_j from each T_{σ_j}, there exists a (unique) global node x such that for all i we have $x^i = \bigcup_j x_j^i$.*

Moreover, for every collective strategy σ, 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 $t_\tau(x) = t_\sigma(f(x))$, where $f(x) = \bigcup_i f_i(x_i)$. 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_τ is a subset of types of T_σ. Then, by Remark 3 we conclude:

Lemma 3. *The collective strategy τ implements ϕ.*

Hence, whenever for a realizable $\phi \in \text{LTL}_\Box$ exist levels l_1 and l_2 with the same type under σ, we can construct a collective strategy τ for which every local strategy τ_i uses only the first l_2 levels of the corresponding σ_i, and Lemma 3 guarantees that τ implements ϕ. By our previous observation and the pigeonhole principle, l_2 is upper bounded by $2^{2^{k|V|}}+1$, and thus every local strategy τ_i operates in the first $2^{2^{k|V|}}+1$ 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 $\overline{S}_n$ of star architectures with pairwise disjoint inputs is decidable in EXPSPACE.*

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

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

V. Synthesis without the next operator

In the current section we consider a fragment of LTL without the $\mathcal{X}$ operator, for which the problem of realizability is decidable in non-deterministic exponential time in the size of the specification.

Fragment LTL$_{\text{AG}}$. We consider LTL$_{\text{AG}}$ that consists of formulas ϕ from the following LTL fragment:

$$\begin{aligned}\phi &= \bigwedge_i \Box P_i \rightarrow \left(\bigwedge_i \Box Q_i \wedge \bigwedge_i \Box\Diamond R_i \wedge \bigwedge_i \Diamond F_i \right) \\ &\equiv \Box \bigwedge_i P_i \rightarrow \left(\Box \bigwedge_i Q_i \wedge \bigwedge_i \Box\Diamond R_i \wedge \bigwedge_i \Diamond F_i \right) \\ &\equiv \Box P \rightarrow \left(\Box Q \wedge \bigwedge_i \Box\Diamond R_i \wedge \bigwedge_i \Diamond F_i \right)\end{aligned}$$

for $i \in \{1, \dots m\}$, 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$_{\text{AG}}$ can express specification that conists of conjunction of safety assumptions, and guarantees where each guarantee is a safety, reachability, or a liveness condition.

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

Proof: The right to left direction is immediate. For the left to right direction, assume that ψ is realizable, and let σ_i be the strategy of process $p_i \in \mathcal{P}$ in $\mathcal{A}$. Construct strategies τ_i such that $\tau_i(x) = \sigma_i(z)$ for every $x = zy \in \left(2^{I(p_i)}\right)$ and $z \in 2^{I(p_i)}$, and let τ be the collective strategy of τ_i. For every infinite path in T, we have that $\ell_\tau(\pi)[i] = \ell_\sigma(\pi)[1]$ for all $i \in \mathcal{N}$. Since σ implements ψ, it follows that $\ell_\sigma(\pi)[1] \models Q$. Hence for all i, we have $\ell_\tau(\pi)[i] \models Q$, and thus $\ell_\tau(\pi) \models \psi$, and all t_i are memoryless strategies. ∎

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

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

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

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

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

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 σ^{R_i} implements ϕ_{R_i}, and the collective strategy σ^{F_i} implements ϕ_{F_i}. Construct local strategies τ_j such that for every $x = yz$ with $|z| = (1 + |x| \mod 2m)$, we have $\tau_j(x) = \sigma_j^{R_{|z|}}(z)$ if $|z| \leq m$, and $\tau_j(x) = \sigma_j^{F_{|z|-m}}(z)$ if $|z| > m$ (i.e. the local strategy τ_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 τ be the collective strategy of all τ_j and consider an arbitrary path π in T. Either $\ell_\tau(\pi)[k] \models \neg P$ for some k, or for all k, it holds $\ell_\tau(\pi)[k] \models P$, and by construction, for $i = 1 + k \mod 2m$, we have $\ell_\tau(\pi)[k] \models Q \wedge R_i$ when $i \leq m$ and $\ell_\tau(\pi)[k] \models Q \wedge F_{i-m}$ when $i > m$. In both cases, $\ell_\tau(\pi) \models \phi$.
(ii) For the left to right direction, assume that for some i, ϕ_{R_i} is not realizable (the analysis is similar for ϕ_{F_i}). By Lemma 5, $\Diamond(P \rightarrow (Q \wedge R_i)$ is not realizable. Hence, for any collective strategy σ there exists some path π in T_σ, such that for all k, we have $\ell_\sigma(\pi)[k] \models P \wedge (\neg Q \vee \neg R_i)$, and σ does not implement ϕ. ∎

Hence, Lemma 6 establishes that every formula $\phi \in$ LTL$_{\text{AG}}$ is realizable if and only if it admits local strategies for all the corresponding ϕ_i, by providing a constructive argument. As a consequence of Lemma 4, deciding whether every ϕ_i is realizable reduces to realizing the propositional formula $(P \rightarrow (Q \wedge R_i \wedge F_i)$. This can be done in 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 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 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_i depends on x_i, 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 σ_1, σ_2 the Skolem functions).

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

Proof: Consider any DQBF formula ψ : $\forall x_1 \dots \forall x_k \exists y_1(\overrightarrow{x_1}) \dots \exists y_n(\overrightarrow{x_n}) \varphi(x_1, \dots x_k, y_1 \dots y_n)$ 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 $\overrightarrow{x_i}$. We construct the architecture $\mathcal{A} = (\mathcal{P}, p_e, V, E)$, where $\mathcal{P}$ contains $n+1$ processes, $V = \{x_i \in \psi\} \cup \{y_i \in \psi\}$, process p_i receives as inputs from the environment all $\overrightarrow{x}_i$, outputs variable y_i, while the environment uses all remaining x_j as hidden variables. We construct the specification $\phi = \Box\varphi \in$ LTL$_{\text{AG}}$. Both $\mathcal{A}$ and ϕ are polynomial in the size of ψ. Because of Lemma 4,

ϕ is realizable in $\mathcal{A}$ iff φ is realizable in $\mathcal{A}$. In turn, φ is realizable iff ψ is valid, with local strategies σ_i corresponding to the Skolem functions in the functional form of ψ, and universal variables corresponding to all possible choices of the environment in $\mathcal{A}$. Since DQBF validity is NEXPTIME-hard [20], the statement follows. ■

Hence, we have the following result.

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

Observe that Lemma 6 reduces the problem of realizability of some $\varphi \in$ LTL$_{\text{AG}}$ to realizing a set of formulas of the form $\Box Q$, 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$_{\text{AG}}$, 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 exmaples.

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

References

[1] A. Church, "Logic, arithmetic and automata," in *Proceedings of the international congress of mathematicians*, pp. 23–35, 1962.

[2] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[3] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," POPL '89, pp. 179–190, ACM, 1989.

[4] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," SFCS '90, pp. 746–757 vol.2, 1990.

[5] A. Pnueli, "The temporal logic of programs," in *FOCS*, pp. 46–57, 1977.

[6] J. H. Reif, "Universal games of incomplete information," STOC '79, pp. 288–308, ACM, 1979.

[7] G. L. Peterson and J. H. Reif, "Multiple-person alternation," in *FOCS*, pp. 348–363, 1979.

[8] O. Kupferman and M. Y. Vardi, "Synthesizing distributed systems," in *LICS*, pp. 389–398, 2001.

[9] S. Mohalik and I. Walukiewicz, "Distributed games," in *FSTTCS*, pp. 338–351, 2003.

[10] B. Finkbeiner and S. Schewe, "Uniform distributed synthesis," LICS, pp. 321–330, 2005.

[11] R. Alur, S. La Torre, and P. Madhusudan, "Playing games with boxes and diamonds," in *CONCUR*, pp. 127–141, 2003.

[12] R. Alur and S. La Torre, "Deterministic generators and games for ltl fragments," *ACM Trans. Comput. Log.*, vol. 5, no. 1, pp. 1–25, 2004.

[13] J. Kretínský and J. Esparza, "Deterministic automata for the (f, g)-fragment of ltl," in *CAV*, pp. 7–22, 2012.

[14] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *VMCAI*, LNCS 3855, Springer, pp. 364–380, 2006.

[15] Y. Godhal, K. Chatterjee, and T. A. Henzinger, "Synthesis of AMBA AHB from formal specification: A case study," *STTT*, 2011.

[16] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer, "Interactive presentation: Automatic hardware synthesis from specifications: a case study," in *DATE*, pp. 1188–1193, 2007.

[17] C. Papadimitriou, *Computational complexity*. Addison-Wesley, 1994.

[18] O. Kupferman, M. Y. Vardi, and P. Wolper, "An automata-theoretic approach to branching-time model checking," *Journal of the ACM (JACM)*, vol. 47, no. 2, pp. 312–360, 2000.

[19] W. J. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *JCSS*, vol. 4, no. 2, pp. 177 – 192, 1970.

[20] G. Peterson, J. Reif, and S. Azhar, "Lower bounds for multiplayer non-cooperative games of incomplete information," *Journal of Computers and Mathematics with Applications*, vol. 41, pp. 957–992, 2001.

[21] A. Fröhlich, G. Kovásznai, and A. Biere, "A dpll algorithm for solving dqbf," *Pragmatics of SAT*, vol. 2012, 2012.