Information Flow Guided Synthesis with Unbounded Communication*

Bernd Finkbeiner[1], Niklas Metzger[1(✉)], and Yoram Moses[2]

[1] CISPA Helmholtz Center for Information Security, Saarland, Germany
{finkbeiner, niklas.metzger}@cispa.de
[2] The Andrew and Erna Viterbi Faculty of Electrical and Computer Engineering and the Taub Faculty of Computer Science, Technion, Israel
moses@technion.ac.il

Abstract. Information flow guided synthesis is a compositional approach to the automated construction of distributed systems where the assumptions between the components are captured as information-flow requirements. Information-flow requirements are hyperproperties that ensure that if a component needs to act on certain information that is only available in other components, then this information will be passed to the component. We present a new method for the automatic construction of information flow assumptions from specifications given as temporal safety properties. The new method is the first approach to handle situations where the required amount of information is unbounded. For example, we can analyze communication protocols that transmit a stream of messages in a potentially infinite loop. We show that component implementations can then, in principle, be constructed from the information flow requirements using a synthesis tool for hyperproperties. We additionally present a more practical synthesis technique that constructs the components using efficient methods for standard synthesis from trace properties. We have implemented the technique in the prototype tool FlowSy, which outperforms previous approaches to distributed synthesis on several benchmarks.

1 Introduction

More than 65 years after its introduction by Alonzo Church [7], the synthesis of reactive systems, and especially the synthesis of *distributed* reactive systems, is still a most intriguing challenge. In the basic reactive synthesis problem, we translate a specification, given as a formula in a temporal logic, into an implementation that is guaranteed to satisfy the specification for every possible input from the environment. In the synthesis of *distributed systems* [32], we must find an implementation that consists of multiple components that communicate with each other via shared variables in a given architecture. While the basic synthesis

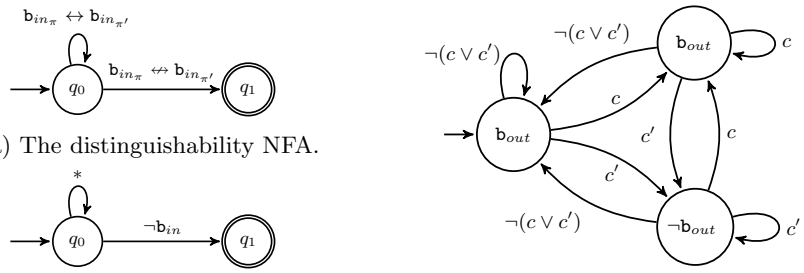
* This work was funded by the German Israeli Foundation (GIF) Grant No. I-1513-407./2019, by DFG grant 389792660 as part of TRR 248 – CPEC, and by the ERC Grant HYPER (No. 101055412).

problem is, by now, well-supported with algorithms and tools (cf. [5, 23]), and despite a long history of theoretical advances [15, 25, 27, 28, 30, 32], no practical methods are currently known for the synthesis of distributed systems.

A potentially game-changing idea is to synthesize the systems compositionally, one component at a time [4, 6, 14, 18, 24, 26, 33]. The key difficulty in automating compositional synthesis is to find assumptions on the behavior of each component that are sufficiently strong so that each component can guarantee the satisfaction of the specification based on the guarantees of the other components, and, at the same time, sufficiently weak, so that the assumptions can actually be realized. In our previous work on *information flow guided synthesis* [17], we identified situations in which certain components must act on information that these components cannot immediately observe, but must instead obtain from other components. Such situations are formalized as information-flow assumptions, which are hyperproperties that express that the component eventually receives this information. Once the information flow assumptions are known, the synthesis proceeds by constructing the components individually so that they satisfy the information-flow assumptions of the other components provided that their own information-flow assumptions are likewise taken care of.

Technically, the synthesis algorithm identifies a finite number of sets of infinite sequences of external inputs, so-called *information classes*, such that the component only needs to know the information class, but not the individual input trace. In the first step, the output behavior of the component is fixed based on an abstract input that communicates the information class to the component. This abstract implementation is called a *hyper implementation* because it leaves open how the information is encoded in the actual inputs of the component. Once all components have hyper implementations, the abstract input is then replaced by the actual input by inserting a monitor automaton that derives the information class from the input received by the component.

This approach has two major limitations. The first is that the information flow requirement only states that the information will *eventually* be transmitted. This is sufficient for liveness properties where the necessary action can be delayed until the information is received. For safety, however, such a delay may result in a violation of the specification. As a result, the information flow assumptions of [17] are insufficient for handling safety, and *the compositional synthesis approach is thus limited to liveness specifications.* The second limitation is due to the restriction to a *finite* number of information classes. As a result, the compositional synthesis approach is only successful if a solution exists that *acts on just a finite amount of information.* The two limitations severely reduce the applicability of the synthesis method. Most specifications contain a combination of safety and liveness properties (cf. [23]). While it is possible to effectively approximate liveness properties through *bounded* liveness properties (cf. [20]), which are safety properties, the converse is not true. Likewise, most distributed systems of interest are reactive in the sense that they maintain an ongoing interaction with the external environment. As a result, they do not conform to the limitation that they only act on a finite amount of information. For example, a communi-



(a) The distinguishability NFA.

(b) An information class NFA.

(c) The safety hyper implementation.

Fig. 1: The prefix distinguishability of the sequence transmission protocol as NFA in (a). The NFA representing the information class for output $\mathtt{b}_{out}$ is shown in (b), where (c) is a hyper implementation of the receiver on information classes.

cation protocol receives a new piece of information in each message and is hence required to transmit an unbounded amount of information.

In this paper, we overcome both limitations with a new method for information flow guided synthesis that handles both safety properties and specifications of tasks that require the communication of an unbounded amount of information. In order to reason about safety, we consider *finite* prefixes of external inputs rather than infinite sequences. The key idea is to collect sets of finite sequences *of the same length* into information classes. Such an information class refers to a specific point in time (corresponding to the length of its traces) and identifies the information that is needed at this point in time to avoid a violation of the safety property. We then only require that the number of information classes is finite *at each point in time*, while the *total* number of information classes over the infinitely many prefixes of an execution may well be infinite. This allows us to handle situations where again and again some information must be transmitted in a potentially infinite loop.

2 Running Example: Sequence Transmission

Our running example is a distributed system that implements sequence transmission. The system consists of two components, the *transmitter* t and the *receiver* r. At every time step, the transmitter observes the current input bit $\mathtt{b}_{in}$ from the external environment, the transmitter can communicate via $\mathtt{c}_b$ with the receiver, and the receiver controls the output $\mathtt{b}_{out}$. To implement a sequence transmission protocol, the receiver must output the value of the input bit one time step after it is received by the transmitter. We can state this specification using the LTL formula $\square(\mathtt{b}_{in} \leftrightarrow \bigcirc \mathtt{b}_{out})$ for the receiver, and assume the transmitter specification to be *true*. In this example, compositional synthesis is only possible with assumptions about the communication between the components. We utilize an *information-flow assumption* for compositional synthesis specified in the HyperLTL formula $\forall\pi\forall\pi'.(\mathtt{c}_{b_\pi} \leftrightarrow \mathtt{c}_{b_{\pi'}})\,\mathcal{U}(\mathtt{b}_{in_\pi} \not\leftrightarrow \mathtt{b}_{in_{\pi'}} \wedge \mathtt{c}_{b_\pi} \not\leftrightarrow \mathtt{c}_{b_{\pi'}})$.

The formula states that on any pair of traces π and π' of an implementation, the communication bit $\mathtt{c}_b$ on both traces must be equivalent until there is a difference on the input bit $\mathtt{b}_{in}$ as well as a difference on the communication bit $\mathtt{c}_b$. This implies that whenever the receiver must distinguish two input traces, it will observe a difference on its local inputs, namely $\mathtt{b}_{in}$. A nondeterministic finite automaton (NFA) accepting all finite traces that must be distinguished at the same time point is depicted in Figure 1a. In the course of this paper, we show that, for safety properties, the distinguishability requirement yields an information-flow assumption specified over finite traces. Based on the assumption, we heuristically build information classes over finite traces, such that all finite traces in the same class do not need to be distinguished. Figure 1b shows an NFA for one of the two information classes. It accepts all finite traces that have $\neg\mathtt{b}_{in}$ in the last step. On all these traces, the output $\neg\mathtt{b}_{out}$ is correct. For this example, there is only one other information class, namely the finite traces with $\mathtt{b}_{in}$ in the last step. We use the information classes to synthesize a *hyper implementation* for the receiver, depicted in Figure 1c. A hyper implementation receives the current information classes, which are c and c' on the transitions, as input, and outputs the local outputs of the component. Whenever c is the input, the correct output for all traces in c must be set by the receiver. Note that, in this example, c and c' cannot occur together as there is no common output for $\mathtt{b}_{in} \wedge \neg\mathtt{b}_{in}$. The hyper implementation is correct for all transmitter implementations. After synthesizing both hyper implementations, for the transmitter and the receiver, we compose and decompose them to obtain local implementations. Throughout this paper, we first define the prefix distinguishability and prefix information-flow assumption. We then build assume and guarantee specifications that, based on the information classes, guarantee the correctness of the hyper implementations, and finally, we show how to construct the local solutions to complete the synthesis procedure.

3 Preliminaries

Architectures. In this paper, we consider distributed architectures with two components: p and q. Such architectures are given as tuple $(I_p, I_q, O_p, O_q, O_e)$, where I_p, I_q, O_p, O_q, and O_e are all subsets of the set $\mathcal{V}$ of boolean variables. O_p and O_q are the sets of *output variables* of p and q. We denote by O_e the output variables of the uncontrollable external environment. We refer to O_e also as the *external inputs* of the system. O_p, O_q and O_e form a partition of $\mathcal{V}$. Finally, I_p and I_q are the *input variables* of components p and q, respectively. The inputs and outputs are disjoint, i.e., $I_p \cap O_p = \emptyset$ and $I_q \cap O_q = \emptyset$. Each of the inputs I_p and I_q of the components is either an output of the environment or an output of the other component, i.e., $I_p \subseteq O_q \cup O_e$ and $I_q \subseteq O_p \cup O_e$. For a set $V \subseteq \mathcal{V}$, every subset $V' \subseteq V$ defines a *valuation* of V, where the variables in V' have value *true* and the variables in $V \setminus V'$ have value *false*.

Implementations. For a set of atomic propositions AP divided into inputs I and outputs O, with $I \cap O = \emptyset$, a 2^O-labeled 2^I-transition system is a 4-tuple

(T, t_0, τ, o), where T is a set of states, $t_0 \in T$ is an initial state, $\tau : T \times 2^I \to T$ is a transition function, and $o : T \to 2^O$ is a labeling function. An implementation of an architecture $(I_p, I_q, O_p, O_q, O_e)$ is a pair (T_p, T_q), consisting of T_p, a 2^{O_p}-labeled 2^{I_p} transition system T_p, and T_q, a 2^{O_q}-labeled 2^{I_q} transition system T_q. The *composition* $T = T_p||T_q$ of the two transition systems $(T^p, t_0^p, \tau^p, o^p)$ and $(T^q, t_0^q, \tau^q, o^q)$ is the $2^{O_p \cup O_q}$-labeled 2^{O_e}-transition system (T, t_0, τ, o), where $T = T^p \times T^q, t_0 = (t_0^p, t_0^q), \tau((t^p, t^q), x) = (\tau^p(t^p, (x \cup o^q(t^q)) \cap I_p), \tau^q(t^q, (x \cup o^p(t^p)) \cap I_q))$, $o(t^p, t^q) = o^p(t^p) \cup o^q(t^q)$, where $x \in 2^{O_e}$.

Specifications. The specifications are defined over the variables $\mathcal{V}$. For a set $V \subseteq \mathcal{V}$ of variables, a *trace* over V is an infinite sequence $x_0x_1x_2 \ldots \in (2^V)^\omega$ of valuations of V. A *specification* over $\mathcal{V}$ is a set $\varphi \subseteq (2^\mathcal{V})^\omega$ of traces over $\mathcal{V}$. Two traces over disjoint sets $V, V' \subset \mathcal{V}$ can be *combined* by forming the union of their valuations at each position, i.e., $x_0x_1x_2 \ldots \sqcup y_0y_1y_2 \ldots = (x_0 \cup y_0)(x_1 \cup y_1)(x_2 \cup y_2) \ldots$. Likewise, the *projection* of a trace onto a set of variables $V' \subseteq \mathcal{V}$ is formed by intersecting the valuations with V' at each position: $x_0x_1x_2 \ldots \downarrow_{V'} = (x_0 \cap V')(x_1 \cap V')(x_2 \cap V') \ldots$. For a trace π we use $\pi[n]$ to access the set on π at time step n, and $\pi[n \ldots m]$ for the interval of π from index n to m. Our specification language is linear-time temporal logic (LTL) [31] with the set $\mathcal{V}$ of variables serving as the atomic propositions. We use the usual Boolean operations, the temporal operators Next $\bigcirc$, Until $\mathcal{U}$, Globally $\Box$, and Eventually $\Diamond$, and the semantic evaluation of (finite) traces π with $\pi \vDash \varphi$. LTL formulas can be represented by nondeterministic Büchi automata (NBAs) with an exponential blow-up. A finite trace $\pi \in (2^\mathcal{V})^*$ is a bad prefix of an LTL formula φ if $\pi \nvDash \varphi$ and $\pi \cdot \pi' \nvDash \varphi$ for all $\pi' \in (2^\mathcal{V})^\omega$. An LTL formula is a *safety* formula if every violation has a bad prefix. Specifications over architectures are conjunctions $\varphi_p \wedge \varphi_q$ of two LTL formulas, where φ_p is defined over $O_p \cup O_e$, i.e., φ_p relates outputs of the component p to the outputs of the environment, and φ_q is defined over $O_q \cup O_e$. We call these specifications the *local* specifications of the component. An initial run $T(i_0, i_1, \ldots) = t_0t_1 \ldots \in T^\omega$ for an infinite sequence of inputs $i_0, i_1 \ldots \in 2^{O_e}$ is an infinite sequence of states produced by the transition function such that $t_i = \tau(t_{i-1}, i_{i-1})$ for all $i \in \mathbb{N}$ and t_0 is the initial state. The set of traces $\mathit{Traces}(T)$ of an implementation $T = (T^p, T^q)$ is then defined as all $(o(t_0) \cup i_0)(o(t_1) \cup i_1) \ldots \in (2^\mathcal{V})^\omega$ where $T(i_oi_1 \ldots) = t_0t_1 \ldots$ for some $i_oi_1i_2 \ldots \in (2^{O_e})^\omega$. An implementation *satisfies* a specification φ if the traces of the implementation are contained in the specification, i.e., $\mathit{Traces}(T^p, T^q) \subseteq \varphi$. Given an architecture and a specification φ, the synthesis problem is to find an implementation $T = (T_p, T_q)$ that satisfies φ. We say that a specification φ is *realizable* in a given architecture if such an implementation exists, and *unrealizable* if not.

Automata. A non-deterministic automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, q_o, \delta, F)$ where Σ is the input alphabet, Q is a set of states, q_o is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and F is a set of accepting states. For an input word $\sigma_0\sigma_1 \ldots \sigma_k \in \Sigma^k$, a finite word automaton (NFA) $\mathcal{F}$ accepts a finite run $q_0q_1 \ldots q_k \in Q^k$ where $q_i \in \delta(q_{i-1}, \sigma_{i-1})$, if $q_k \in F$. A Büchi automaton (NBA) $\mathcal{A}$ accepts all infinite runs $q_0q_1 \ldots \in Q^\omega$ that visit states in F infinitely often. An

automaton is deterministic if the transition function δ is injective. The language of an automaton $\mathcal{A}$ is the set of its accepting runs, and is denoted by $\mathcal{L}(\mathcal{A})$.

Hyperproperties. Information-flow assumptions are hyperproperties. A *hyperproperty over* $\mathcal{V}$ is a set $H \subseteq 2^{(2^{\mathcal{V}})^\omega}$ of sets of traces over $\mathcal{V}$ [9]. An implementation (T_p, T_q) satisfies the hyperproperty H iff the set of its traces is an element of H, i.e., $\mathit{Traces}(T_p, T_q) \in H$. A convenient specification language for hyperproperties is the temporal logic HyperLTL [8], which extends LTL with trace quantification, i.e., $\forall\pi.\varphi$ and $\exists\pi.\varphi$. In HyperLTL, atomic propositions are indexed by a trace variables, which make expressing properties like "ψ *must hold on all traces*"possible, expressed by $\forall\pi.\ \psi$. Dually, one can express that "*there exists a trace on which* ψ *holds*", denoted by $\exists\pi.\ \psi$. Sometimes, a hyperproperty can be expressed as a binary relation on traces. A relation $R \subseteq (2^{\mathcal{V}})^\omega \times (2^{\mathcal{V}})^\omega$ of pairs of traces defines the hyperproperty H, where a set T of traces is an element of H iff for all pairs $\pi, \pi' \in T$ of traces in T it holds that $(\pi, \pi') \in R$. We call a hyperproperty defined in this way a *2-hyperproperty*. In HyperLTL, 2-hyperproperties are expressed as formulas with two universal quantifiers and no existential quantifiers. A 2-hyperproperty can equivalently be represented as a set of infinite sequences over the product alphabet $\mathcal{V}^2$: we can represent a given 2-hyperproperty $R \subseteq \mathcal{V}^\omega \times \mathcal{V}^\omega$, by $R' = \{(\sigma_0, \sigma'_0)(\sigma_1, \sigma'_1)\ldots \mid (\sigma_0\sigma_1\ldots, \sigma'_0\sigma'_1\ldots) \in R\}$. This representation is convenient for the use of automata to recognize 2-hyperproperties.

4 Prefix Information Flow

As argued in [17], identifying information flow between the components is crucial for distributed synthesis, because the specification may require a component's actions to depend on external inputs that are not directly observable by the component. To react to the external inputs correctly, at least the relevant information must be transferred to the component. The fundamental concept to identify when a component requires information transfer is captured by a *distinguishability* relation on sequences of environment outputs. We recall the definition of distinguishability for a component p from [17]:

Definition 1 (Trace distinguishability [17]). *Let φ_p be an LTL specification of p. The corresponding* trace distinguishability relation *is defined as*

$$\tau_p = \{(\pi_e, \pi'_e) \in (2^{O_e})^\omega \times (2^{O_e})^\omega \mid \forall \pi_p \in (2^{O_p})^\omega.\, \pi_e \sqcup \pi_p \not\models \varphi_p \text{ or } \pi'_e \sqcup \pi_p \not\models \varphi_p\}$$

The trace distinguishability relation is defined w.r.t. pairs of infinite traces, where each trace records all outputs of the environment, building up all the information that is presented to the system. Two traces are related iff there exists no infinite trace of p's outputs that satisfies the specification for both (environment) input traces. For example, the traces in the sequence transmission protocol are related by τ_r if they differ on $\mathtt{b}_{in}$ at least once. We now turn the distinguishability relation into an assumption for the component. On traces related

by τ_p, the component must observe a difference in its *local* inputs, namely the set I_p. The relation itself only considers infinite traces over all variables that are *not outputs* of the single component, independent of the architecture. Therefore, the information-flow assumption (IFA) built from the distinguishability relation enforces that on all related (environment input) traces, there is a difference on the component's input:

Definition 2 (Trace information-flow assumption [17]). *Let τ_p be the trace distinguishability relation for p. The* information flow assumption *$\mathcal{I}_p$ is the 2-hyperproperty defined by the relation*

$$R_{\mathcal{I}_p} = \{(\pi, \pi') \in (2^{\mathcal{V}})^\omega \times (2^{\mathcal{V}})^\omega \mid \text{ if } (\pi{\downarrow_{O_e}}, \pi'{\downarrow_{O_e}}) \in \tau_p \text{ then } \pi{\downarrow_{I_p}} \neq \pi'{\downarrow_{I_p}}\}$$

The trace information-flow assumption is necessary for a component p; every implementation of the distributed system will satisfy the information-flow assumption from [17]. In its generality, this definition specifies that the values of the local inputs to p have to be different *at some time point*, without an explicit or implicit deadline. This is critical in two ways: On the one hand, liveness specifications, as in the example $\mathtt{b}_{in} \leftrightarrow \Diamond \mathtt{b}_{out}$, will never determine an explicit point in time where the information must be present. On the other hand, safety specifications always include a fixed deadline for the reaction of the component, which, if the information is not present, cannot be met. This deadline, however, is not accounted for in the information-flow assumption, and an algorithm cannot rely on availability of the information during synthesis.

In [17] we solve the liveness issue by introducing a time-bounded information-flow assumption. The time bound acts as a placeholder for the exact time point of information flow. The locally synthesized receiver must then be correct for all such possible time points. Because of the arbitrary deadline, the assumptions cannot suffice to find a solution for a safety specification of the receiver either; they are too weak. We solve this issue by restricting the attention to safety specifications. Consider, for example, the safety property $\varphi_r = \square(\mathtt{b}_{in} \leftrightarrow \bigcirc \mathtt{b}_{out})$ of our running example. To satisfy this property, the receiver r must observe the value of $\mathtt{b}_{in}$ on its local inputs in exactly one time step, otherwise, it cannot react to $\mathtt{b}_{in}$ in time. With this observation, we can state a stronger distinguishability relation over pairs of *finite* traces.

Definition 3 (Prefix distinguishability). *Let φ_p be the safety specification for component p. The* prefix distinguishability relation *is defined as*

$$\begin{aligned}\rho_{\varphi_p} = \{(\pi, \pi') \in (2^{O_e})^m \times (2^{O_e})^m, m \in \mathbb{N} \mid \forall \pi_p \in (2^{O_p})^m.\\ \pi \sqcup \pi_p \not\models_m \varphi_p \text{ or } \pi' \sqcup \pi_p \not\models_m \varphi_p\\ \text{and } \forall n \in \mathbb{N}, n < m.\ \exists \pi'_p \in (2^{O_p})^n.\\ \pi[0 \dots n] \sqcup \pi'_p \models_n \varphi_p \text{ and } \pi'[0 \dots n] \sqcup \pi'_p \models_n \varphi_p\}\end{aligned}$$

The first condition states that, for the two related input traces of length m, the specification is violated for all possible output sequences for p of the same

length. The second condition enforces that m is the first position at which the trace pair must be distinguished, i.e., for all previous positions of the traces, there exists a common output sequence that satisfies the specification on both traces. Every violation of a safety specification has a *minimal bad prefix* [11], and hence every violation that originates in the indistinguishability of two traces is captured by Definition 3. For liveness specifications, no two traces are related by this definition: One can inductively reason that for every $(\pi, \pi') \in \tau_{\varphi_p}$ this pair of traces is not in ρ_{φ_p}, i.e., $(\pi, \pi') \notin \rho_{\varphi_p}$, since for every chosen m, one can find an output trace of p that violates the formula after time point m.

Prefix distinguishability is the core concept of our synthesis method. We now show that we can build an automaton that accepts a pair of finite environment output traces iff they are related. We say that an automaton $\mathcal{A}$ *recognizes* a relation R if $\mathcal{L}(\mathcal{A}) = R$.

Theorem 1. *For a component p with specification φ_p, there exists a non-deterministic finite automaton with a doubly exponential number of states in the length of φ_p that recognizes the prefix distinguishability relation ρ_{φ_p}.*

Proof. We construct a non-deterministic finite automaton (NFA) $\mathcal{F}$ that accepts precisely all pairs of traces over $(2^{O_e})^m \times (2^{O_e})^m$, where $m \in \mathbb{N}$, that are related by ρ_{φ_p}. Let φ'_p be the formula φ_p where all atomic propositions $a \in AP$ are renamed to a', and let $\mathcal{V}'$ be a set containing a copy v' of every variable $v \in \mathcal{V}$. We build the NBA $\mathcal{B} = \mathcal{A}_{\varphi_p} \times \mathcal{A}_{\varphi'_p}$, where $\mathcal{A}_{\varphi_p}$ and $\mathcal{A}_{\varphi'}$ are constructed with a standard LTL-to-NBA translation respectively, and the operator $\times$ builds the product of two NBAs. $\mathcal{B}$ now accepts all tuples of traces that each satisfy φ_p. Let $\mathcal{C}$ be the NBA that restricts the transition relation of $\mathcal{B}$ s.t. edges are only present if the output variables of p are equal $\bigwedge_{o \in O_p} o \leftrightarrow o'$ holds, enforcing that both traces agree on the output while satisfying the specification. We now existentially project to the set $O_e \cup O'_e$ to build $\mathcal{D}$, whose alphabet does not contain the component's outputs. To accept the pairs of traces that do not satisfy the formula, we negate $\mathcal{D}$, denoted by $\bar{\mathcal{D}}$. In the last step of the construction, we transform the NBA $\bar{\mathcal{D}}$ to an NFA $\mathcal{F}$ using the *emptiness per state* construction of [3]. This yields an NFA that accepts the prefix distinguishability relation. The size of the automaton is doubly exponential in the size of the formula. The first exponent stems from the LTL to NBA construction, and the second from negating the automaton $\mathcal{F}$. □

Similar to Definition 2, we now turn the safety distinguishability relation into an information-flow assumption that must be guaranteed by the component that observes the respective environment output. The assumptions include specific information-flow deadlines for pairs of traces at which the component must observe the information at the latest. The information-flow assumption, again, is a 2-hyperproperty enforcing that pairs of traces that are related by the prefix distinguishability relation have an observable difference for the component.

Definition 4 (Prefix information-flow assumptions). *Let ρ_{φ_p} be the prefix distinguishability relation for p. The corresponding* prefix information flow

assumption $\mathcal{P}_p$ *is the 2-hyperproperty defined by the relation*

$$R_{\mathcal{P}_p} = \{(\pi, \pi') \in (2^{\mathcal{V}})^\omega \times (2^{\mathcal{V}})^\omega \mid \text{ if } \exists m \in \mathbb{N} \text{ s.t. } (\pi[0 \dots m], \pi'[0 \dots m]]) \in \rho_{\varphi_p} \\ \text{then } \pi{\downarrow_{I_p}}[0 \dots m-1] \neq \pi'{\downarrow_{I_p}}[0 \dots m-1]\}$$

On all finite trace pairs in the prefix distinguishability relation ρ_{φ_p}, there must be a difference on I_p before the deadline m. Restricting the observable difference to happen before the *deadline* m is crucial for the receiving component. Whereas the prefix distinguishability relation relates finite traces, the prefix information-flow assumption is a hyperproperty over infinite traces. Unsurprisingly, every implementation of a distributed system satisfying safety LTL specifications satisfies the corresponding prefix information-flow assumption.

Lemma 1. *The prefix information-flow assumption is necessary for safety LTL specifications.*

Proof. Assume that there exists an implementation (T_p, T_q) satisfying the safety LTL specifications φ_p and φ_q but not $\mathcal{P}_p$ and $\mathcal{P}_q$. Since $\mathcal{P}_p$ is not satisfied, there exists a pair of traces π, π' such that $(\pi{\downarrow_{O_e}}[0 \dots m], \pi'{\downarrow_{O_e}}[0 \dots m]) \in \rho_{\varphi_p}$ and $\pi{\downarrow_{I_p}}[0 \dots m+1] = \pi'{\downarrow_{I_p}}[0 \dots m+1]$. The deterministic system must therefore choose the same output for the timestep $m+1$ since the inputs are the same. This contradicts the assumption: either $\pi[0 \dots m+1]$ or $\pi'[0 \dots m+1]$ is a minimal bad prefix since, otherwise, the traces would not be related by the prefix distinguishability relation. □

We are now ready to return to the sequence transmission example. The prefix distinguishability automaton for $\square(\mathtt{b}_{in} \leftrightarrow \bigcirc \mathtt{b}_{out})$ is depicted in Figure 1a. The automaton accepts a 2-hyperproperty whose alphabet is a pair of valuations of $\mathtt{b}_{in}$. Note that the communication bit from t to r is not restricted by the prefix distinguishability. The automaton terminates whenever a sequence of inputs must be distinguished. For example, starting in the initial state, the input words $\mathtt{b}_{in}$ on π and $\neg\mathtt{b}_{in}$ on π' lead immediately to an accepting state; these finite traces need to be distinguished. However, if $\mathtt{b}_{in}$ is equivalent on both traces, the automaton stays in the initial non-accepting state. By abuse of notation, we use X_p for X_{φ_p}, e.g., ρ_p for ρ_{φ_p}, if φ_p is clear from context.

The automata for the prefix distinguishability and the prefix information-flow assumption can be very complex; even if two traces are different at point n, it can be decided at position $n+m$ if the difference of the inputs results in a necessary information flow, and the automaton might need to store the observed difference during all m intermediate steps. We evaluate the size of the prefix distinguishability automaton empirically in Section 7. With the prefix information-flow assumption, we could construct a hyperproperty synthesis problem similar to [17]. In practice, however, synthesis from hyperproperties is largely infeasible, because it hardly scales to more than a few system states [16]. In the following, we show that this problem can be avoided by reducing the compositional synthesis problem to the much more practical synthesis from trace properties.

5 Unbounded Communication in Distributed Systems

Computing the information flow between the components in a distributed system, as shown in Section 4, is the first step for compositional synthesis. In the second and more complex step, the synthesis procedure needs to guarantee (1) that the component that observes the information actually transmits the information, and (2) that the component requiring the information correctly assumes the reception. We construct an *assume specification*, which ensures that the component correctly assumes the information flow, and a *guarantee specification*, which enforces the correct transmission of information.

5.1 Receiving Information

A component cannot realize its specification only based on its local observations; it needs to assume that the required information is transmitted during execution. The prefix information-flow assumption is one class of necessary assumptions, i.e., every transmitter implementation must satisfy it, and the hyperproperty can be assumed without losing potential solutions. In many cases, this assumption is also sufficient; if the receiver assumes this exact information flow, the local synthesis problem is realizable. During synthesis, we do not know what actual information the component currently has. The synthesis procedure only has *partial information* of all environment outputs. Which information is actually transmitted at which time point is finally decided by the synthesis process of the *transmitter*. However, the receiver's implementation must be correct for every possible information in every step. We, therefore, collect all traces at a position that do not need to be distinguished by component p at time n, i.e., there exists a prefix of p's outputs that works on all traces.

Definition 5 (Prefix information class). *Let ρ_p be the prefix distinguishability relation for p. The information class of a trace π at position $n \in \mathbb{N}$ is the set of traces $[\pi]_p^n = (2^{O_e})^n \backslash \{\pi' \in (2^{O_e})^n \mid (\pi, \pi') \in \rho_p\}$*

We now construct a trace property that, given an information class c^n, enforces that the output by the component is correct for all traces in the information class c^n. This property specifies exactly one step of outputs, namely $n+1$. Since we consider safety LTL properties, it is sufficient to incrementally specify the outputs according to the satisfaction of the LTL formula.

Definition 6 (Information class specification). *Let φ_p be the LTL specification for component p, $n \in \mathbb{N}$, and let c_n be a prefix information class at position $n-1$. The information class specification $\mathbb{C}_p^n \subseteq (2^{\mathcal{V} \backslash O_q})^\omega$ is defined as*

$$\mathbb{C}_p^n = \{\pi_e \sqcup \pi_o \mid \pi_e \in (2^{\mathcal{V} \backslash O_p})^n, \pi_o \in (2^{O_p})^n \\ \text{s.t. } \forall \pi_e' \in c_{n-1}.\pi_e'[0 \dots n] \sqcup \pi_o[0 \dots n] \vDash_n \varphi_p\}.$$

The output traces in $\mathbb{C}_p^n$ need to be correct for every environment output trace that is in the information class. Here, if an environment output trace is

not in the information class, we do not restrict any behavior. We now introduce a crucial assumption: That the number of information classes over all time steps is *bounded*. In general, this is not necessary: one can distinguish every trace from every other trace, such that the information classes increase in every time step. However, if the number of information classes is bounded, we present an effective heuristic for constructing them on the prefix distinguishability assumption in Section 5.1. Each information class c (which is now *not* parametric in the time point) is then a set of finite traces $(2^{O_e})^\star$, which is exactly the set of traces in each step that do not need to be distinguished by a component. Consider, for example, the sequence transmission specification $\varphi = \square(\mathtt{b}_{in} \leftrightarrow \bigcirc \mathtt{b}_{out})$. The information classes w.r.t. Definition 5 are all traces that are equal on the environment outputs up to time-point $n-1$. This builds infinitely many information flow classes. It is, however, possible to reduce the information classes to a finite representation. In our example, it is sufficient to check for the previous position of the traces: all finite traces that are equal at $n-1$ do not need to be distinguished. This yields two information classes, one for $\mathtt{b}_{in}$ at the previous step and one for $\neg\mathtt{b}_{in}$ at the previous step. The NFA accepting one of them is depicted in Figure 1b. With the assumption that we are given a finite set of information classes as subsets of $(2^{O_e})^*$, we are able to build an *assume specification*, which assumes that information classes are received if necessary, and can react to any possible *consistent* sequence of information classes. The information classes $\mathcal{C}$ are now part of the alphabet for the input traces and we use c for refering to a specific information class and as an atomic proposition.

Definition 7 (Assume specification). *Let φ_p be the component specification and $\mathcal{C}$ be the finite set of information classes, where each $c \in \mathcal{C}$ is a subset of $(2^{O_e})^*$. The trace property $\mathbb{A} \subseteq (\mathcal{C} \cup 2^{O_p})^\omega$ is defined as*

$$\mathbb{A}_p^{\mathcal{C}} = \{\pi_{\mathcal{C}} \cup \pi_o \mid \pi_{\mathcal{C}} \in \mathcal{C}^\omega, \pi_o \in (2^{O_p})^\omega, \forall n \in \mathbb{N}. \forall c \in \pi_{\mathcal{C}}[n-1]. \\ \forall \pi_e[0 \dots n-1] \in c.\ \textit{if } \pi_c \textit{ is consistent, then } \pi_e \sqcup \pi_o[0 \dots n] \vDash_n \varphi_p\},$$

where a finite prefix $\pi_{\mathcal{C}} \in \mathcal{C}^n$ is consistent if it holds that for all $0 \leq m < n$, all finite traces in $\pi_e[0 \dots m]$ have a prefix in $\pi_e[0 \dots m-1]$.

The assume specification collects, for a sequence of information classes, all component outputs that are correct for all environment outputs in this information class. The consistency of input traces specifies the correct reveal of information classes. It cannot be the case that a trace that was distinguishable from the current trace in step $n-1$ is indistinguishable in n. Note that a correct transmitter will implement only consistent traces. Let's assume we are given the information classes $\mathcal{C} = \{c, c'\}$ for the sequence transmission problem, where $c = (\{\mathtt{b}_{in}\}, \{\neg\mathtt{b}_{in}\})^*\{\mathtt{b}_{in}\}$ and $c' = (\{\mathtt{b}_{in}\}, \{\neg\mathtt{b}_{in}\})^*\{\neg\mathtt{b}_{in}\}$. These classes suffice to implement the receiver: whenever the trace over $\mathcal{C}^n$ ends in c, the receiver has to respond with $\mathtt{b}_{out}$ and it should respond with $\neg\mathtt{b}_{out}$ whenever the trace ends in c'. Each information class c can be split into the information classes c_n by fixing the length of the traces to n.

Lemma 2. *Let $\mathcal{C}_p$ be the finite set of information classes for component p. Every implementation satisfying the assume specification $\mathbb{A}_p^{\mathcal{C}}$ also satisfies the information class specification $\mathbb{C}_p^n$ for all $n \in \mathbb{N}$ and $c \in \mathcal{C}_p$.*

This lemma follows directly from the definition of the assume specification: It collects all information class specifications for the given set of information classes. Note that correctness is only specified for the set of information classes, not the information flow assumption. If the information classes are not total, in the sense that all distinguished traces are in one of the classes, then the receiver is not correct for *all* implementations of the sender.

5.2 Transmitting Information

While a transmitter has to satisfy its local specification, it must also guarantee that the information flow that the receiver relies on is transmitted in time. In general, this is, again, a hyperproperty synthesis problem: The combination of the local specification of q and the prefix information-flow assumption of p is the 2-hyperproperty that the implementation of q needs to satisfy. However, we propose a framework for more involved (incomplete) trace property synthesis algorithms, potentially speeding up the transmitter synthesis significantly. In contrast to the receiver, the transmitter of information can choose the synthesis strategy; As long as the transmitter satisfies the information-flow assumption, the receiver will assume this implementation as feasible and can react to the information flow correctly during composition. We specify a class of trace properties s.t. each element specifies a subset of the implementations that satisfy a correct transmitter.

Definition 8 (Guarantee specification). *Let p and q be components and $\mathcal{I}\varphi_p$ be the IFA for φ_p. The set $\mathbb{G}_{\rho_p} \subseteq (2^{I_q \cup O_q})^\omega$ is a guarantee specification if all 2^{O_q}-labeled 2^{I_q}-transition systems that satisfy $\mathbb{G}$ also satisfy $\mathcal{I}_p$.*

The first crucial difference between the guarantee specification and the assume specification in Definition 7 is that the transmitter must guarantee a difference on the traces in ρ_p whereas the receiver can only assume to observe a difference whenever ρ_p relates two traces. Additionally, the guarantee specification can specify a subset of implementations of all possible transmitters. We show this difference in the following example: Consider our running example specification $\varphi = \square(\mathtt{b}_{in} \leftrightarrow \bigcirc \mathtt{b}_{out})$. One of the (infinitely) many guarantee specifications can be the set of traces specified by the LTL formula $\square(\mathtt{b}_{in} \leftrightarrow \neg \mathtt{c}_b)$, which enforces that every $\mathtt{b}_{in}$ is communicated to the receiver by setting $\mathtt{c}_b$ to *false*.

It remains to show that we can construct guarantee specifications for prefix information-flow assumptions effectively. We will highlight two useful guarantee specifications, one that is implemented in our prototype and one that utilizes the information classes. We begin with the *full-information specification*. It forces the transmitter to send, if possible, all information and therefore reduces the distributed synthesis problem to monolithic synthesis. This concept was already

presented in [32] where it was called adequate connectivity and later extended by Gastin et al. [21].

Definition 9 (Full-information specification). *Let p and q be components, and $f : O_e \cap I_q \to O_q \cap I_p$ be a bijection. The* full-information specifciation *for q is the trace property*

$$\begin{aligned}\mathbb{F}_p = \{\pi_e \sqcup \pi_o \mid \pi_e \in (2^{O_e \cap I_q})^\omega, \pi_p \in (2^{O_q \cap I_p})^\omega, \forall v \in (O_e \cap I_q),\\ \textit{either } \forall n \in \mathbb{N}. v \in \pi_e[n] \textit{ iff } f(v) \in \pi_o[n+1]\\ \textit{or } \forall n \in \mathbb{N}. v \in \pi_e[n] \textit{ iff } f(v) \notin \pi_o[n+1]\}\end{aligned}$$

This specification forces the sender to assign exactly one value of a communication variable to every input variable. This choice must hold for every point in time and can not be changed, ensuring that every input combination is uniquely represented by the communication variables. The full-information specification is a guarantee specification for every possible information-flow assumption. Since *every* input bit is guaranteed to be transmitted, every different input trace can be distinguished, not only the ones required to be distinguished by the prefix distinguishability relation. The full-information specification is a sufficient condition for realizing the sender; if there is an implementation for satisfying $\mathbb{F}$, then this implementation is a correct sender. It is not a necessary specification, the sender might be able to encode the inputs to a smaller set of communication variables. The second guarantee specification is based on the information classes.

Definition 10 (Information Class Guarantee). *Let $\mathcal{C}'_p$ be the finite set of information classes of p projected to the inputs of q, s.t. the information classes $c \in \mathcal{C}'_p$ are subsets of $(2^{O_e \cap I_q})^*$. Let furthermore $f : \mathcal{C} \to 2^{O_q \cap I_p}$ be a bijection. The information class guarantee $\mathbb{I}^{\mathcal{C}}_q \subseteq (2^{(O_e \cap I_q) \cup (O_q \cap I_p))})^\omega$ is defined as*

$$\begin{aligned}\mathbb{I}^{\mathcal{C}}_q = \{\pi_e \cup \pi_o \mid \pi_e \in (2^{O_e \cap I_q})^\omega, \pi_o \in (2^{O_q \cap I_p})^\omega, \forall n \in \mathbb{N}, \forall c \in \mathcal{C}'_p\\ \textit{if } \pi_e[0 \dots n] \in c \textit{ then } f(c) \in \pi_o[n+1]\}.\end{aligned}$$

The specification tracks, for an environment output trace π_e, the current information class. Whenever the finite trace is in an information class c, the transmitter must set the combination of its outputs to the values as specified by the bijection f. The receiver p can therefore observe c by decoding the outputs of q on $O_q \cap I_p$. Similar to the assume specification, the correctness of the information class guarantee depends on the information classes:

Lemma 3. *If a set of information classes $\mathcal{C}$ is sufficient to synthesize φ_p then $\mathbb{I}^{\mathcal{C}}_q$ is a guarantee specification for φ_p.*

If providing the information classes at every step is not sufficient for synthesis, then either the specification is unrealizable or at least one information class falsely contains two traces that need to be distinguished. The assume and guarantee specifications in Section 5 build the foundation for synthesizing local

components that satisfy the local specification and the information-flow assumption. In most distributed systems, however, components are not solely receivers nor transmitters, but both simultaneously. We now define local implementations that are correct w.r.t. information classes, called safety hyper implementations.

5.3 Safety Hyper Implementations

Hyper implementations were introduced in [17] specifying local implementations of a distributed system that are correct for all possible implementations of all other components. The hyper implementations observe all inputs of the environment but are forced to react to them only if necessary, without restricting the possible solution space of other components. For example, the implementation of the receiver r in the sequence transmission protocol is a 2^{O_r}-labeled 2^{I_r}-transition system, but any locally synthesized solution for r must react to inputs only observed by t. We use the *information classes* of Definition 5 to specify and synthesize a different approach to hyper implementations. Recall that we assume a bounded number of information classes $\mathcal{C}$.

Definition 11 (Safety hyper implementation). *Let p and q be components, e be the environment, and $\mathcal{C}_p$ be a set of information classes. A safety hyper implementation $\mathcal{H}_p$ of p is a 2^{O_p}-labelled $\mathcal{C}_p \cup 2^{I_p}$-transition system.*

The safety hyper implementation branches over the information classes and the local inputs to p and reacts with local outputs. The safety hyper implementation of our running example is depicted in Figure 1c. Compared to (non-safety) hyper implementations in [17], the safety hyper implementations do not contain a special input variable `t` that signalizes the reception of information. This deadline is explicitly present in the prefix distinguishability relation and can be computed on the automaton representing the prefix distinguishability relation. Since we consider safety properties, there always exists a pre-determined time frame between the environment input and the necessary reception of the information - a fact that we utilize heavily during hyper implementation construction. We now formalize when a safety hyper implementation is correct.

Definition 12 (Correctness of safety hyper implementation). *Let p, q, and e be the components of a distributed system and the environment, and φ_p, φ_q be the local specifications. A* safety hyper implementation *$\mathcal{H}_p$ is correct if it implements $\mathbb{A}_{\varphi_p}$ and some $\mathbb{G}_{\varphi_q}$.*

Correct hyper implementations of p are compatible with all correct implementations of q, i.e., all possible sequences of information provided by *some* transmitter, and implement *one* solution to the information-flow assumption of q. Since assume and guarantee specifications are trace properties, we can synthesize safety hyper implementations with trace property synthesis algorithms once the Büchi automata for the specifications are constructed.

Fig. 2: The steps in the algorithm for compositional synthesis with prefix information flow assumptions.

6 Synthesis with Prefix Information Flow Assumptions

In this section, we present algorithms for generating assume and guarantee specifications, the synthesis of hyper implementations, and obtaining the solutions for each component. Combined, this builds our compositional synthesis approach with information-flow assumptions for distributed systems.

6.1 Automata for Assume and Guarantee Specifications

The first step in our synthesis approach is to construct the assume specification which builds on a finite set of information classes. According to Definition 5, there is, in theory, an unbounded number of information classes. Our Algorithm 1 therefore iteratively builds automata that accept, for each prefix length, one information class. Given the automaton for the prefix distinguishability relation over $\Sigma_\varphi \times \Sigma_{\varphi'}$, the function identicalAPs returns an automaton $\mathcal{A}_{id}$ that accepts exactly one input trace over the alphabet Σ_φ at each time step. This is achieved by choosing one explicit proposition combination for each edge in the automaton.

On this automaton, the function call allTraces($\mathcal{A}_{id} \cap \overline{\mathcal{A}_c}$) collects all traces that do not need to be distinguished from $\mathcal{A}_{id}$. These are the traces in the negation of the prefix distinguishability relation that are related to $\mathcal{A}_{id}$. The function allTraces can be computed by renaming the primed propositions on the edges of the automaton. This concludes the computation of the first information class. The algorithm continues by removing $\mathcal{A}_{id}$ from the prefix distinguishability automaton and computing the next information class until the current automaton for the prefix distinguishability relation is empty.

Algorithm 1: Information Classes

```
1 let informationClasses(A_ρ):=
2   let A_c = A_ρ
3   let A_id = identicalAPs(A_ρ)
4   let result = ∅
5   while L(A_id) ≠ ∅ do
6     result.add(allTraces(A_id ∩ ¬A_c))
7     A_c = A_ρ ∩ ¬A_id
8     A_id = identicalAPs(A_c)
9   return result
```

Algorithm 1 yields, if it terminates, n finite automata $\mathcal{F}_c$ where all traces in each $\mathcal{F}_c$ do not need to be distinguished. This implies that there exists a common output combination for each time-step that is correct for each trace

in the automaton. We now show a construction for the assume specification in Definition 7.

Construction 1. *We first transform the n finite automata $\mathcal{F}_1, \dots, \mathcal{F}_n$ for the information classes, as obtained as the result of Algorithm 1, to the respective information class specification (see Definition 6). For each automaton, we build the intersection of the goal automaton $\mathcal{A}_\varphi$ and $\mathcal{F}_i$. The resulting automaton accepts all traces in the information class with outputs as specified by φ. This yields an NBA $\mathcal{B}$ that only accepts a subset of all input traces. We lift it to an automaton for the information class specification by unionizing all input and output combinations that do not occur on $\mathcal{F}_i$, which is $\mathcal{A}_{true} \backslash \mathcal{B}$, where $\mathcal{A}_{true}$ is the automaton accepting all input and output combinations. After performing this construction for all n information class automata, the intersection of all of them accepts the assume specification.*

We use this automaton for the local synthesis of each component. The local specification is implicitly satisfied by the hyper implementation of the assume specification since it is encoded in the construction. We now show how to construct the full information specification in Definition 9.

Construction 2. *Let $I = I_q \cap O_e$ be the inputs observed by q and $O = O_q \cap I_p$. We assume that $|I| \leq |O|$ since we can only transmit all information if we have at least as many communication variables as environment output variables. Let $f : I \to O$ be a bijection that maps input variables to output variables. We construct the LTL formula $\varphi = \bigwedge_{i \in I} \square(i \leftrightarrow \bigcirc f(i)) \vee \square(i \leftrightarrow \bigcirc \neg f(i))$. This formula enforces that, for every $i \in I$, either the value of i is copied to $f(i)$ at every point in time, or the negation of i's value is copied to $f(i)$ at every point. The corresponding automaton whose language is a full-information specification is $\mathcal{A}_\varphi$, obtained by a standard LTL to NBA translation.*

Together with a guarantee specification, the hyper implementation satisfies its own local specification and the guarantee of the other component.

6.2 Compositional Synthesis

The last step in the compositional synthesis algorithm is the composition and decomposition of the hyper implementations. After this process, we obtain the local implementations of the components and therefore the implementation of the distributed system. During composition and decomposition, we need to replace the information class variables with the actual locally received input. The composition collects all environment and component outputs, as well as the information classes for both components. This includes unreachable states, namely combinations of information classes and environment outputs that are impossible (the finite environment output trace is not in the information class). We eliminate these states in Definition 14. The composition is defined as follows:

Definition 13 (Composition). *Let p, q be components and $\mathcal{H}_p = (T^p, t_o^p, \tau^p, o^p)$ and $\mathcal{H}_q = (T^q, t_0^q, \tau^q, o^q)$ be their respective safety hyper implementations.*

The composition $\mathcal{H} = \mathcal{H}_p || \mathcal{H}_q$ *is a* $2^{O_p \cup O_q}$*-labeled* $2^{O_e} \cup \mathcal{C}_p \cup \mathcal{C}_q$*-transition system* (T, t_p, τ, o)*, where* $T = T^p \times T^q$*,* $t_0 = (t_0^p, t_0^q)$*,* $o((t^p, t^q)) = o^p(t^p) \cup o^q(t^p)$*, and*

$$\tau((t^p, t^q), x) = \ \tau^p(t^p, (x \cup o^q(t^q)) \cap (I_p \cup \mathcal{C}_p)), \tau^q(t^q, (x \cup o^p(t^p)) \cap (I_q \cup \mathcal{C}_q))$$

The state space is the cross product of the hyper implementations and the labeling function is the union of the local hyper implementation's outputs. The transition function ensures that the global inputs over $2^{O_e} \cup \mathcal{C}_p \cup \mathcal{C}_q$ are separated into the inputs of the respective hyper implementations, namely $\mathcal{C}_p \cup I_p$ and $\mathcal{C}_q \cup I_q$. For every state in the cross-product, the composition branches for every environment output and information class to a local state of a component. Some of these states are unreachable. For our running example, the composition includes a transition with $\neg \mathtt{b}_{in}, c'$, even though the trace with $\neg \mathtt{b}_{in}$ in the last step cannot be in c'. We now filter states according to consistency. We consider $\mathcal{H}(x)$ as the hyper implementation $\mathcal{H}$ terminating in x.

Definition 14 (Filter). *Let* $\mathcal{H} = (T, t_0, \tau, o)$ *be the composition of the* 2^{O_p}*-labeled* $2^{\mathcal{C}_p \cup I_p}$*-transition system* $\mathcal{H}_p$ *and the* 2^{O_q}*-labeled* $2^{\mathcal{C}_q \cup I_q}$*-transition system* $\mathcal{H}_q$*. The consistent composition of* $\mathcal{H}_p$ *and* $\mathcal{H}_q$ *is the hyper implementation* $\mathcal{H}' = (T', t_0', \tau', o')$*, with* $T' = T$*,* $t_0' = t_0$*,* $o' = o$*, and*

$$\tau'((t^p, t^q), x) = \begin{cases} \tau((t^p, t^q), x) & \text{if } \forall c \in x. \mathcal{H}(t^p, t^q) \subseteq \mathcal{L}(\mathcal{F}_c) \\ \emptyset & \text{else} \end{cases}$$

A finite trace π of length n over $2^{O_e} \cup \mathcal{C}_p \cup \mathcal{C}_q$ is impossible to reach if c is in $\pi[n]$ but $\pi \downarrow_{O_e}$ is not in the information class represented by c. Computing if a state is unreachable includes language inclusion of the subsystem terminating in the state and the automaton of the information class. However, an algorithm that enforces consistency can monitor the current information class of a state during a forward traversal of the composed hyper implementations. In the next and final step, the decomposition then projects the composition to only the *observable* outputs of a component. For some input combinations, this yields a set of reachable states, of which we choose one for the decomposition. In essence, all these states are viable successors for the current input combination.

Definition 15 (Decomposition). *Let* $\mathcal{H} = (T, t_p, \tau, o)$ *be the consistent composition of the* 2^{O_p}*-labeled* $2^{\mathcal{C}_p \cup I_p}$*-transition system* $\mathcal{H}_p$ *and the* 2^{O_q}*-labeled* $2^{\mathcal{C}_q \cup I_q}$*-transition system* $\mathcal{H}_q$*. Furthermore, let min be a function returning the minimal element for a subset of* T *w.r.t. some total ordering over the states of* T*. The decomposition* $\mathcal{H}|_p$ *is a* 2^{O_p}*-labelled* 2^{I_p}*-transition system* $(T^p, t_0^p, \tau^p, o^p)$ *where* $T^p = T$*,* $t_0^p = t_0$*,* $o^p((t^p, t^q)) = o((t^p, t^q)) \cap O_p$*, and*

$$\tau^p(t, x) = min\{t' \mid \exists y \in 2^{(O_e \cup \mathcal{C}_p) \setminus I_p}. t' = \tau(t, x \cup y)\}$$

The full compositional synthesis algorithm is shown in Figure 2. Given the two local specifications, the first step is computing the prefix distinguishability NFAs. Based on those, the assume specifications and guarantee specifications for both components are constructed and build the inputs to the local synthesis

procedures. Note that the guarantee specification can be any strategy that implements the information flow assumption, e.g., any scheduling paradigm. After intersecting the two automata, the components must satisfy the assume and the guarantee specification together, which is achieved by trace property synthesis on the intersection of the automata. The problem is unrealizable if either the prefix information-flow assumption is not sufficient for synthesis (there could be necessary behavioral assumptions), or not all information can be communicated to the receiver. After composition, consistency, and decomposition, the algorithm terminates with two local implementations that, together, implement a correct distributed system:

Corollary 1. *Let p and q be components with local specifications φ_p and φ_q. The distributed system implementation returned by the algorithm depicted in Figure 2 satisfies the local specifications.*

7 Experiments

We implemented the compositional synthesis algorithm described so far in our prototype called FlowSy. The implementation builds on the popular infinite word automaton manipulation tool spot [13] for translation, conversion, and emptiness checking of NBAs. FlowSy implements the support for the finite automata, the construction of prefix distinguishability in Construction 4, the construction of the information classes in Algorithm 1, and building automata for the assume specification in Construction 1 and full information specification Construction 2. The synthesis of the hyper implementations is performed by converting the Büchi automata to deterministic parity games and solving them with the solver oink [12]. We report on two research questions, (1) how do the prefix distinguishability automaton and the information classes scale w.r.t. formula size and information flow over time and (2) how does FlowSy compare to the existing bounded synthesis approach for distributed system HyperBosy presented in [16]. Note that, at the time of evaluation, HyperBosy was the only tool for distributed synthesis that we were able to compare against. A comparison with the existing information flow guided synthesis algorithm with bounded communication in [17] is infeasible since the supported languages of input specifications are disjoint. All experiments are run on a 2.8 GHz processor with 16 GB RAM, the timeout was 600 seconds, and the results are shown in Table 1.

Benchmarks. The benchmarks scale in 3 different dimensions: the number of independent variables, time-steps in between information reception and corresponding output, and combinatorics over input and output variables. The first one is independent communication of n input variables in *sequence transmission*. This parametric version of the running example has n conjuncted subformulas of the form $\square(i \leftrightarrow \bigcirc o)$. For the *delay* benchmark, the number of variables is constant, but the number of time steps between input and output is increased, i.e., the formulas have the form $\square(i \leftrightarrow \bigcirc^n o)$. The last two benchmarks build

Table 1: This table summarizes the experimental results. The Benchmark and Parameter columns specify the current instance. The columns $|\varphi|$, $|\rho|$, and $|\mathcal{C}|$ give the size of the formula, the number of states in the prefix distinguishability automaton, and the number of information classes, respectively. The last two columns report the running time of FlowSy and BosyHyper in seconds.

Benchmark	**Par.**	$\|\varphi\|$	$\|\rho\|$	$\|\mathcal{C}\|$	**FlowSy**	**BosyHyper**
Delay	1	5	4	2	1.74	0.97
	2	6	8	2	**1.87**	TO
	3	7	16	2	**1.84**	TO
	4	8	32	2	**1.94**	TO
	5	9	64	2	**2.36**	TO
Sequence Transmission	1	5	4	2	1.83	**1.42**
	2	11	6	4	**5.28**	TO
	3	16	10	8	**36.81**	TO
Conjunctions	1	5	4	2	3.18	**0.92**
	2	9	4	4	**4.35**	91.80
	3	13	4	8	**9.20**	TO
	4	17	4	16	TO	TO
Disjunctions	1	5	4	2	**3.25**	6.26
	2	9	4	4	**5.63**	60.08
	3	13	4	8	**12.14**	TO
	4	17	4	16	TO	TO

Boolean combinations over the inputs. The *conjunctions* benchmark enforces that the conjunctions over the inputs are mirrored in the outputs. *Disjunctions* is constructed in the same way but with disjunctions in between variables. Formulas are $\square(i_1 \wedge i_1 \wedge \ldots \leftrightarrow \bigcirc o_1 \wedge o_2 \wedge \ldots)$ and $\square(i_1 \vee i_1 \vee \ldots \leftrightarrow \bigcirc o_1 \vee o_2 \vee \ldots)$.

Scaling. FlowSy primarily scales in the number of computed information classes. Most interestingly, for benchmark *delay*, the number of information classes is constantly 2, even though the size of the prefix distinguishability automaton grows exponentially. Independent of the length of the current trace, the automaton for the information class checks that the current position is equal to the position n steps earlier. This can indeed be represented by two information classes. For synthesizing the conjunction and disjunction benchmarks, the situation is reversed. Even though the prefix distinguishability automaton is constant, the number of information classes grows exponentially in the parameter, collecting all possible combinations of input variables. For the sequence transmission benchmark, all reported values scale with the input parameter, which leads to an expected increase of the running time until the timeout at step 4 (not included in Table 1).

Comparison to BosyHyper. FlowSy clearly outscales BosyHyper. Most interestingly, the delay benchmark shows the almost constant running time for

FlowSy. Since the number of information classes stays the same, the synthesis of the hyperproperties only scales for transmitting the information. BosyHyper must store all values for all $\bigcirc^n$ steps during synthesis, which immediately increases the search space to an infeasible size. For the benchmarks conjunction and disjunction, one can observe that, although the information classes scale exponentially, the running time of FlowSy is significantly faster than that of BosyHyper, which is already at 91 seconds for parameter 2. In summary, the compositionality of FlowSy is always beneficial for the synthesis process and it saves on the execution time dramatically when the complex communication in the distributed system can be reduced to a small number of information classes.

8 Related Work

Compositional synthesis for monolithic systems, i.e., architectures with one component and the environment, is a well-studied field in reactive synthesis, for example in [14,18,24,26] and most recently in [1]. In multi-component systems with partial observation, compositionality has the potential to improve algorithms significantly, for example in reactive controller synthesis [2, 22]. Assume-guarantee synthesis adheres to the same synthesis paradigm as our approach: the local components infer assumptions over the other components to achieve the local goals [4, 6]. The assumptions are *trace properties*, restricting the behavior of the components which often is not necessary. If the assumptions are not sufficient, i.e., too weak to locally guarantee the specification, the assumption can be iteratively refined [29]. Another approach is weakening the acceptance condition to dominance [10] or certificates that specify partial behavior of the components in an iterative fashion [19]. In our previous work on information flow guided synthesis [17], we have introduced the concept of compositional synthesis with information-flow assumptions. The work presented in the paper overcomes the two major limitations of this original approach, namely the limitation to liveness (or, more precisely, co-safety properties) and the limitation to specifications that can be realized by acting only on a a finite amount of information.

9 Conclusion

We have presented a new method for the compositional synthesis of distributed systems from temporal specifications. Our method is the first approach to handle situations where the required amount of information is unbounded. While the information-flow assumptions are hyperproperties, we have shown that standard efficient synthesis methods for trace properties can be utilized for the construction of the components. In future work, we plan to study the integration of the information-flow assumptions computed by our approach with the assumptions on the functional behavior of the components generated by techniques from behavioral assume-guarantee synthesis [4, 6]. Such an integration will allow for the synthesis of systems where the components collaborate both on the distribution and on the processing of the distributed information.

References

1. Akshay, S., Basa, E., Chakraborty, S., Fried, D.: On dependent variables in reactive synthesis (2024)
2. Alur, R., Moarref, S., Topcu, U.: Compositional synthesis of reactive controllers for multi-agent systems. In: Chaudhuri, S., Farzan, A. (eds.) CAV (2016). https://doi.org/10.1007/978-3-319-41540-6_14
3. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. (2011). https://doi.org/10.1145/2000799.2000800
4. Bloem, R., Chatterjee, K., Jacobs, S., Könighofer, R.: Assume-guarantee synthesis for concurrent reactive programs with partial information. In: Baier, C., Tinelli, C. (eds.) TACAS 2015, Proceedings (2015). https://doi.org/10.1007/978-3-662-46681-0_50
5. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_27
6. Chatterjee, K., Henzinger, T.A.: Assume-guarantee synthesis. In: Grumberg, O., Huth, M. (eds.) TACAS 2007, Proceedings (2007). https://doi.org/10.1007/978-3-540-71209-1_21
7. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic. vol. 1, pp. 3–50. Cornell Univ., Ithaca, NY (1957)
8. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) POST 2014 (2014). https://doi.org/10.1007/978-3-642-54792-8_15
9. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security
10. Damm, W., Finkbeiner, B.: Automatic compositional synthesis of distributed systems. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014. Proceedings (2014). https://doi.org/10.1007/978-3-319-06410-9_13
11. d'Amorim, M., Rosu, G.: Efficient monitoring of omega-languages. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings (2005). https://doi.org/10.1007/11513988_36
12. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: Beyer, D., Huisman, M. (eds.) TACAS 2018 (2018). https://doi.org/10.1007/978-3-319-89960-2_16
13. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: (CAV'22) (2022). https://doi.org/10.1007/978-3-031-13188-2_9
14. Filiot, E., Jin, N., Raskin, J.: Compositional algorithms for LTL synthesis. In: Bouajjani, A., Chin, W. (eds.) ATVA 2010. Proceedings (2010). https://doi.org/10.1007/978-3-642-15643-4_10
15. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: LICS (2005)
16. Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesizing reactive systems from hyperproperties. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018, Proceedings, Part I (2018). https://doi.org/10.1007/978-3-319-96145-3_16
17. Finkbeiner, B., Metzger, N., Moses, Y.: Information flow guided synthesis. In: Shoham, S., Vizel, Y. (eds.) CAV 2022, Proceedings, Part II (2022). https://doi.org/10.1007/978-3-031-13188-2_25

18. Finkbeiner, B., Passing, N.: Dependency-based compositional synthesis. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020. Proceedings (2020). https://doi.org/10.1007/978-3-030-59152-6_25
19. Finkbeiner, B., Passing, N.: Compositional synthesis of modular systems. In: Hou, Z., Ganesh, V. (eds.) Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Proceedings (2021). https://doi.org/10.1007/978-3-030-88885-5_20
20. Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer **15**(5-6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z
21. Gastin, P., Sznajder, N., Zeitoun, M.: Distributed synthesis for well-connected architectures. Formal Methods in System Design **34**(3), 215–237 (2009)
22. Hecking-Harbusch, J., Metzger, N.O.: Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In: Chen, Y., Cheng, C., Esparza, J. (eds.) ATVA 2019.Proceedings (2019). https://doi.org/10.1007/978-3-030-31784-3_22
23. Jacobs, S., Pérez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018-2021. CoRR (2022). https://doi.org/10.48550/ARXIV.2206.00251
24. Kugler, H., Segall, I.: Compositional synthesis of reactive systems from live sequence chart specifications. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. Proceedings. https://doi.org/10.1007/978-3-642-00768-2_9
25. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Logic in Computer Science (LICS) (2001)
26. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006, Proceedings (2006). https://doi.org/10.1007/11817963_6
27. Madhusudan, P., Thiagarajan, P.: Distributed controller synthesis for local specifications. In: Proc. ICALP'01. vol. 2076. Springer–Verlag (2001)
28. Madhusudan, P., Thiagarajan, P.: A decidable class of asynchronous distributed controllers. In: Proceedings of Concur'02. vol. 2421, pp. 145–160. Springer–Verlag (2002)
29. Majumdar, R., Mallik, K., Schmuck, A., Zufferey, D.: Assume-guarantee distributed synthesis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (2020). https://doi.org/10.1109/TCAD.2020.3012641
30. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. TOPLAS **6**(1), 68–93 (Jan 1984)
31. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977 (1977). https://doi.org/10.1109/SFCS.1977.32
32. Pnueli, A., Rosner, R.: Distributed Reactive Systems Are Hard to Synthesize. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II. pp. 746–757. IEEE Computer Society (1990). https://doi.org/10.1109/FSCS.1990.89597
33. Schewe, S., Finkbeiner, B.: Semi-automatic distributed synthesis. Int. J. Found. Comput. Sci. **18**(1), 113–138 (2007)