Distributed Synthesis with Incomparable Information

Béatrice Bérard, Serge Haddad,
Mathieu Sassolas, and Marc Zeitoun

Research Report LSV-10-17

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Distributed Synthesis with Incomparable Information

B. Bérard[1⋆ ⋆⋆], S. Haddad[2⋆ ⋆⋆], M. Sassolas[1⋆⋆], M. Zeitoun[2,3⋆]

[1] Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606, Paris, France.
E-mail: {Beatrice.Berard,Mathieu.Sassolas}@lip6.fr
[2] ENS Cachan, LSV, CNRS UMR 8643 & INRIA, Cachan, France.
E-mail: {Serge.Haddad,mz}@lsv.ens-cachan.fr
[3] Univ. Bordeaux, LaBRI, CNRS UMR 5800, Bordeaux, France.

Abstract. Given (1) an architecture defined by processes and communication channels between them or with the environment, and (2) a specification on the messages transmitted over the channels, distributed synthesis aims at deciding existence of local programs, one for each process, that together meet the specification, whatever the environment does. Recent work shows that this problem can be solved when a *linear preorder* sorts the agents w.r.t. the information received from the environment.
In this paper we show a new decidability result in the case where this preorder is broken by the addition of noisy agents embedded in a pipeline architecture. This case cannot be captured by the classical framework. Besides, this architecture makes it possible to model particular security threats, known as covert channels, where two users (the sender and the receiver) manage to communicate via a noisy protocol, and despite incomparable views over the environment.

Keywords: distributed synthesis, games, tree automata, covert channels

1 Introduction

Distributed synthesis. Synthesis aims at automatically implementing a specification, often expressed in some high-level logical language, into a program that fulfills this specification. This problem was first raised by Church [5] for open systems, that is, systems communicating with their environment. In Church's problem, the specification is a monadic second order property constraining the sequence of input/output pairs of the system, and the program is searched as a function whose output at each time may depend on all preceding inputs. Church's problem can be reduced to the nonemptiness problem for tree automata [16], yielding a finite state implementation, if there is one.

The synthesis problem can be parametrized by the type of systems considered, the specification language and the requirements on the programs synthesized. Here, we consider *open*, *distributed*, and *synchronous* systems. As explained, *open* means that the system communicates with its environment. *Distributed* means that the system is split over several components called *processes*, each of which receives part of the input provided by the environment. A communication architecture describes how processes may communicate with each other. Finally, *synchronous* means that there is a global clock: at each clock tick, the environment provides inputs to the processes, computations and communications between the processes are performed, and outputs to the environment are produced. For simplicity, we assume that communications are instantaneous (no delays), and we only work with acyclic architectures. For specifications, we consider branching time properties (*e.g.* CTL^*). Since the system is synchronous, its configuration at any time is the value of all input/output/communication channels, and the specifications express properties of the computation trees of such configurations, produced by the system and the environment.

⋆ Partly supported by project DOTS (ANR-06-SETI-003).
⋆⋆ Partly supported by Région Île-de-France: project CoChaT DIGITEO-2009-27HD.

Related work. In the centralized case, the synthesis problem can be seen as a two-player infinite game between the system and the environment [20], and is 2EXPTIME-complete [12], or as a control problem in a reactive environment [11]. In the distributed case, synthesis is more difficult due to incomplete information. It remains decidable (but non-elementary) for CTL^* properties over pipeline architectures [17,13], shown on Fig. 1, where arrows depict communications or outputs. The decision procedure consists in (1) solving the game for the centralized case, viewing the architecture as a single process, and (2) distributing iteratively the centralized program obtained at stage (1): at each iteration step, one only considers two processes. The first step is illustrated in Fig. 1: it considers process P_1 and an abstract process encapsulating all other processes (shown dashed). Once correct implementations for the abstract process have been characterized (by an automaton), one iterates the procedure for this abstract process, split into P_2, and an abstract process encapsulating P_3 to P_n.

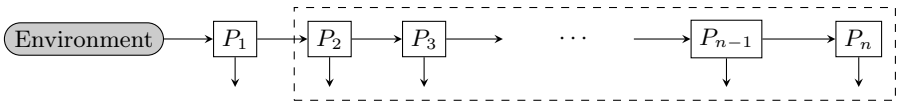


Fig. 1. Pipeline architecture.

The core of the decidability result of [13] relies on constructing, from a specification φ given by a tree automaton $\mathcal{A}_\varphi$, another automaton $\mathcal{A}'_\varphi$ accepting strategies T_R of the abstract process such that there exists a strategy T_S for process P_1 whose composition, $T_S \triangleright T_R$, fulfills the specification φ:

$$T_R \text{ is accepted by } \mathcal{A}'_\varphi \iff \exists T_S,\ T_S \triangleright T_R \text{ is accepted by } \mathcal{A}_\varphi \qquad (\dagger)$$

Results from [17,13] were generalized in [6], which establishes a simple syntactical criterion on the architecture for synthesis to be decidable: decidable architectures are exactly those where the set of processes can be linearly (pre)ordered according to the information from the environment known by each process. Thus, the pipeline is indeed the typical decidable architecture.

Several attempts have been made to extend this decidability result. At a generic level, the game approach in [3] gives a behavioral characterization of distributed games for safety properties that admit finite memory winning strategies. Another line of work consists in restricting the specifications to enlarge the class of decidable architectures. *Local* specifications, for instance, only constrain for each process its own output depending on its own input. For LTL local specifications, synthesis is decidable for a slightly larger class of architectures, called doubly flanked pipelines [14]. On the other hand, if specifications are not allowed to restrict internal communications, then the distributed synthesis problem *always* admits a positive answer for linearly ordered architectures whose channels have enough bandwidth [7]. Embedding adversaries, modeled as nondeterministic whiteboxes (which are processes that cannot be constrained) has been considered in [19]. In general, this model yields undecidability, and one possible solution is to partly perform the distribution by hand. Our work can be seen as a specialization of this framework to obtain decidability results.

Contributions.

(1) We introduce in Section 3 the *noisy pipeline* (Fig. 2), an architecture extending the pipeline from [17]. It is made of n processes $P_1, P_2, \ldots, P_n$, cooperating to fulfill a specification, communicating synchronously through a *noisy* environment modeled by opposing agents $N_2, \ldots, N_n$. At each round, information flows from P_1 to P_n. Process P_i gets a message from N_i (with $N_1 = \text{Env}$,

the "external" environment) and transmits a message to N_{i+1}, which then adds noise to its input, and transmits the resulting message to P_{i+1}, and so on. There is no communication delay. The architecture is fixed, but two parameters come into play:

- the goal of processes $P_1, \ldots, P_n$, specified by a property linking the input from the environment and the messages transmitted by $P_1, \ldots, P_n$;
- the computation power of each agent N_i, *i.e.*, its ability to add noise.

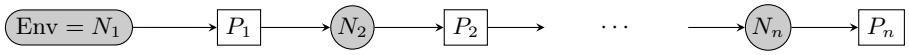


Fig. 2. The *noisy* pipeline

If each N_i is constrained to forward to P_i the very same input it receives from P_{i-1}, then there is actually no noise and we recover the usual pipeline. On the other hand, if this is not the case, the noisy agents may create incomparable information. For this reason, we need to introduce in Section 2 new constructions on trees, completing those of [13].

(2) We show in Section 4.1 that allowing the specification to talk about *all* channels makes synthesis undecidable for $n \geq 2$, by simulating the basic undecidable architecture from [17].

(3) Our main result (Section 4.2) states that distributed synthesis for noisy pipelines is *decidable* if we disallow the specification to constrain the agents N_i. Instead, we require N_i to behave like a *nondeterministic* finite state input/output machine.

Due to the noise produced by the intermediate adversaries N_i, processes $P_1, \ldots, P_n$ have incomparable knowledge w.r.t the environment, so that the aforementioned results do not apply. Our decidability result may even seem surprising, since incomparable information (called *fork* in [6]) leads to undecidability. This is not the case here, because the undecidability result of [6] relies on the ability to specify on each and every channel.

Here, the adversaries N_i, $i > 1$, are themselves distributed, embedded between processes P_i. Each N_i only has a local knowledge, namely the output of P_{i-1}. As in [13], the decidability result relies on an iterative procedure, but the core equation $(*)$, to be explained later, involves an additional quantifier alternation, over the strategy T_N of the noisy adversary (compare to $(\dagger)$):

$$T_R \text{ accepted by } \mathcal{A}'_\varphi \iff \exists T_S \, \forall T_N, \; T_S \triangleright (T_N \blacktriangleright \mathcal{E}_\lambda(T_R)) \text{ accepted by } \mathcal{A}_\varphi \qquad (*)$$

(4) Finally, we apply this result to covert channel analysis (Section 5), an important issue in security. We show how detecting unforeseen information leaks that can be exploited by malicious agents reduces to solving distributed synthesis for a noisy architecture.

2 Constructions on trees

2.1 Words and trees

For an alphabet X, let X^* be the set of finite words on X. In the sequel, X, Y, Z, U are finite alphabets, with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_p\}$, $Z = \{z_1, \ldots, z_\ell\}$ and $U = \{u_1, \ldots, u_m\}$.

A *tree* branching on X and labeled by Y, called a *X-Y-tree*, is a (total) function $T : X^* \to Y$, also called a *labeling* function. We extend the labeling function into a mapping $\hat{T} : X^* \to Y^*$ collecting all labels (except the first one) encountered when reading a word: for $a_1 a_2 \cdots a_k \in X^*$ with $a_i \in X$, we define:

$$\hat{T}(\varepsilon) = \varepsilon \qquad \text{and} \qquad \hat{T}(a_1 a_2 \cdots a_k) = T(a_1) T(a_1 a_2) \cdots T(a_1 a_2 \cdots a_k).$$

A tree is also called a *strategy* for a process reading on alphabet X and writing on alphabet Y. The intuition is that on a path starting from the root, the directions are moves of the first player, while the labels encountered on nodes are the moves of the second player prescribed by the strategy. The *play* according to an X-Y-tree T on input $w \in X^*$ is $T(\varepsilon)\hat{T}(w)$.

Two-layer trees. We introduce *two-layer trees*, a technical tool used to formalize our synthesis problem. Nodes in such a tree are words where letters from X and Y alternate. When the last letter is from X, the node is not labeled (indicated by a $\perp$ symbol). Such a tree is depicted in the middle of Fig. 4. They could be seen as trees branching on $X \times Y$, but the operations (contraction, expansion) are defined more naturally with two-layer trees.

Definition 1. *Let $\perp \notin Z$. A* two-layer tree *branching on X and Y, labeled by Z, also called an $(X;Y)$-Z-*tree*, is a (total) function $T : (XY)^* \cup (XY)^*X \to Z \cup \{\perp\}$, such that $T\big((XY)^*\big) \subseteq Z$ and $T\big((XY)^*X\big) = \{\perp\}$.*

Operations on trees. The *composition* of two trees is standard. For T_1, an X-Y-tree, and T_2, a Y-Z-tree, the composition $T_1 \triangleright T_2$ is the X-$(Y \times Z)$-tree T given by $T(w) = \big(T_1(w), T_2(\hat{T}_1(w))\big)$ for $w \in X^*$. The construction at one level is pictured in Fig. 3 (where the label of the topmost node has been omitted for simplicity). The transformation proceeds inductively top-down: the subtree attached under the node labeled $(y_{\alpha(k)}, z_{\beta(\alpha(k))})$ is the composition of the tree under the node labeled $y_{\alpha(k)}$ in T_1 with the tree under the node labeled $z_{\beta(\alpha(k))}$ in T_2. Note that it differs from the composition in [13] which introduces a delay.

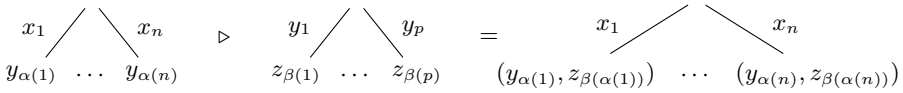


Fig. 3. Composition.

The next technical operations we introduce, pictured in Fig. 4–5, are new. They involve two-layer trees. The first one, called *contraction*, consists in selecting a direction at every node of an even level of a two-layer tree T_2 according to the labels of an ordinary tree T_1.

Definition 2. *Let T_1 be an X-Y-tree and let T_2 be an $(X;Y)$-Z-tree. Define the* contraction $T = T_1 \blacktriangleright T_2$ *of T_2 with respect to T_1 as the following X-Z-tree: for $w = a_1 \cdots a_k \in X^*$, let $\hat{T}_1(w) = b_1 \cdots b_k \in Y^*$. Then $T(w) = T_2(a_1 b_1 \cdots a_k b_k)$.*

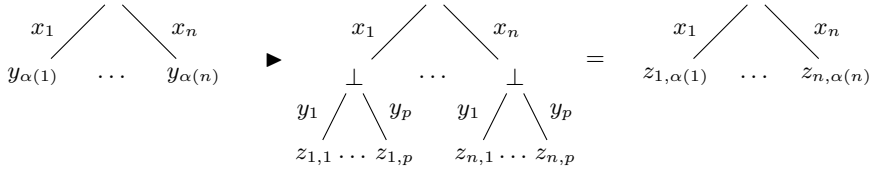
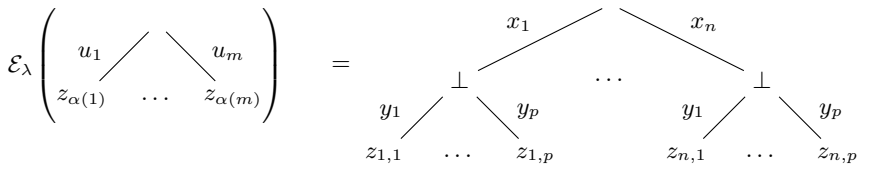


Fig. 4. Contraction.

Again, this operation is an inductive application of a local rewriting, as shown in Fig. 4. The construction proceeds under the node labeled $z_{k,\alpha(k)}$ by combining the subtree of T_1 under the node labeled $y_{\alpha(k)}$ and the subtree of T_2 under the node labeled $z_{k,\alpha(k)}$.



where $z_{i,j} = z_{\alpha(h)}$ with h given by $\lambda(x_i, y_j) = u_h$.

Fig. 5. Expansion w.r.t. function λ.

An *expansion* w.r.t. to a function λ, produces a two-layer tree from an ordinary tree. In the resulting two-layer tree, the label after following successively directions x and y is as the one after following direction $\lambda(x, y)$ in the original (ordinary) tree.

Definition 3. *Let $\lambda : (X \times Y) \to U$, and define $\bar{\lambda} : (XY)^* \to U^*$ as follows: for $w = a_1 b_1 a_2 b_2 \dots a_k b_k \in (XY)^*$, let $\bar{\lambda}(w) = \lambda(a_1, b_1)\lambda(a_2, b_2) \dots \lambda(a_k, b_k)$. The* expansion *of a U-Z-tree T according to λ is the $(X;Y)$-Z-tree $T' = \mathcal{E}_\lambda(T)$ defined by $T'(w) = T(\bar{\lambda}(w))$.*

2.2 Tree automata (TA)

For any finite set Z, let $\mathcal{B}^+(Z)$ be the set of positive propositional formulas on Z, defined by the following grammar (where $\mathbf{t}$ and $\mathbf{f}$ denote true and false, respectively):

$$\varphi ::= \mathbf{t} \mid \mathbf{f} \mid z \in Z \mid \varphi \wedge \varphi \mid \varphi \vee \varphi.$$

A subset Y of Z *satisfies* $\varphi \in \mathcal{B}^+(Z)$, noted $Y \vDash \varphi$, if the truth assignment mapping elements of Y to $\mathbf{t}$ and elements of $Z \setminus Y$ to $\mathbf{f}$ satisfies φ. Any formula of $\mathcal{B}^+(Z)$ can be put in equivalent disjunctive normal form $\bigvee_i \bigwedge_j z_{i,j}$.

A parity *alternating tree automaton* (ATA) on X-Y-trees is given by a tuple $\langle Q, Q^i, \delta, P, \alpha \rangle$ where Q is a set of states, $Q^i \subseteq Q$ is the set of initial states, $\delta : Q \times Y \to \mathcal{B}^+(Q \times X)$ is the transition function, and $\alpha : Q \to P$ is the acceptance condition, with $P = \{0, \dots, p_{\max}\}$ a set of *priorities*.

A *run* ρ of an ATA $\langle Q, Q^i, \delta, P, \alpha \rangle$ on an X-Y-tree T is an unranked tree labeled by $Q \times X^*$ (unranked means here that the number of successors of a node is not constant) such that:
- the root node is labeled by (q, ε) for some state $q \in Q^i$,
- for any node u of ρ labeled by (q, w), there is a subset $S = \{(q_1, a_1), \dots, (q_k, a_k)\}$ of $Q \times X$ such that $S \vDash \delta(q, T(w))$ and u has k successors in ρ labeled by $(q_1, wa_1), \dots, (q_k, wa_k)$.

A run is *accepting* if on all infinite branches, the smallest priority visited infinitely often is even. A tree T is *accepted* by $\mathcal{A}$ if there exists an accepting run on T. We write $\mathcal{L}(\mathcal{A})$ for the set of trees accepted by ATA $\mathcal{A}$. An ATA whose transition relation yields only formulas that can be put in the form $\bigvee_i \bigwedge_{j=1}^{|X|} (q_{i,j}, x_j)$ is called *nondeterministic* (NDTA).

In the case of two-layer trees, the notion of ATA can be adapted by defining the transition function in two parts, one for each layer. Thus w.l.o.g. we assume in the sequel that the states of such an automaton are partionned into two subsets, corresponding to even and odd levels. Furthermore the acceptance condition is specified only on the states of the even levels (the original condition can be recovered with additional finite memory).

Classical results on TA. The class of ATA is closed under complement [15], which is achieved by negating the acceptance condition and syntactically transforming, in transitions, $\bigvee_i \bigwedge_j z_{i,j}$ into

$\bigwedge_i \bigvee_j z_{i,j}$, and exchanging $\mathbf{t}$ and $\mathbf{f}$. The acceptance condition is negated by taking $\overline{P} = \{0, \ldots, p_{\max}+1\}$ and $\overline{\alpha}(q) = \alpha(q) + 1$ for $q \in Q$ as a new acceptance condition. Alternating tree automata can be transformed into nondeterministic TA [15]. The resulting NDTA has a size exponential in the number of priorities of the acceptance condition and in the number of states of the original automaton. It has a number of priorities polynomial in the number of states and in the number of priorities of the original ATA.

3 Architectures with noisy nondeterministic adversaries

We present in this section the new setting of noisy pipelines, an architecture depicted in Fig. 6. It includes noisy agents N_i, who together with the environment, are the adversaries of the processes $P_1, \ldots, P_n$. Given a CTL^* specification over the channels contents, the problem is to decide if there are strategies for processes P_i, so that, whatever the behavior of the environment *and of the noisy adversaries*, the system meets the specification.

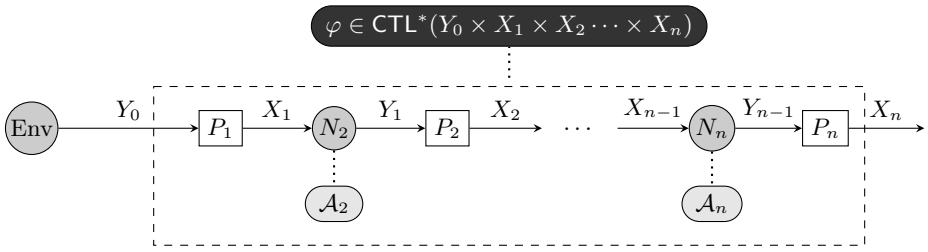


Fig. 6. Noisy architecture with (adversary) noisy agents.

Our main result (Theorem 8 below) states that if the specification *does not* specify the output channels of the noisy agents, synthesis is still decidable, despite incomparable information. The behavior of noisy agents N_i is specified by nondeterministic input/output finite-state machines $\mathcal{A}_i$, the *noisy automata* defined below: upon receiving an input $x \in X$, $\mathcal{A}_i$ nondeterministically changes its state, and then outputs $y \in Y$ according to its new state and input x.

Definition 4. *A* Noisy Automaton *(NA) is a tuple* $\mathcal{A} = \langle X, Y, Q, q_0, \delta, \lambda, F \rangle$ *where*
- X *is the input alphabet,* Y *is the output alphabet,*
- Q *the finite set of states,* q_0 *is the initial state,*
- $\delta : Q \times X \to 2^Q$ *is the transition function,*
- $\lambda : X \times Q \to Y$ *is the output function,*
- $F \subseteq Q$ *is a Büchi acceptance condition.*

Noisy Automata can be seen as a special case of alphabetical transducers [2] where for any state, all incoming transitions on the same letter produce the same output. The Büchi condition allows to restrict the behavior of an adversary more finely, for example by being able to express fairness conditions.

Example. In the NA of Fig. 7, all channels are binary (that is $X = Y = \{0, 1\}$). The noise is specified by two modes $q_=$ and $q_\neq$, one of which is non-deterministically chosen at the beginning of the execution and never changed afterwards. The first mode forwards exactly the received bit, while the second mode sends the opposite of the bit.

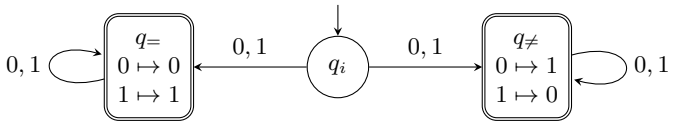


Fig. 7. NA that either always toggles or always copies the input bit.

Definition 5. *Let $\mathcal{A} = \langle X, Y, Q, q_0, \delta, \lambda, F\rangle$ be an NA. A* noise strategy *for $\mathcal{A}$ is an X-Q-tree T such that:*
- *$T(\varepsilon) = q_0$,*
- *for any word $u \in X^*$, and any letter $x \in X$, $T(ux) \in \delta(T(u), x)$,*
- *all branches satisfy the Büchi acceptance condition F of $\mathcal{A}$: some state of F is visited infinitely often.*

While there are only two strategies in the above example, there may be an infinite number of them in general.

In the pipeline architecture (Fig. 6), a strategy of process P_i is a Y_{i-1}-X_i tree T_i. We are looking for the existence of a tuple of strategies $(T_1, \ldots, T_n)$, called a *distributed strategy*, which, combined with *any* tuple $(T_{N_2}, \ldots, T_{N_n})$ of strategies for noisy agents, yields a behavior satisfying φ. The problem is formally stated with tree operations in the next definition.

Definition 6. *Let $(T_1, \ldots, T_n)$ be a tuple of strategies for the processes and $(T_{N_2}, \ldots, T_{N_n})$ be a tuple of noise strategies for the agents. The* outcome *is the Y_0-$(X_1 \times \cdots \times X_n)$-tree T_1^* defined with a backward induction below:*

$$T_n^* = T_n \qquad \text{and} \qquad \text{for } i = n-1, \ldots, 1, \quad T_i^* = T_i \triangleright (T_{N_{i+1}} \blacktriangleright \mathcal{E}_{\lambda_{i+1}}(T_{i+1}^*))$$

where $\lambda_i : X_{i-1} \times Q_i \to Y_i$ is the output function of $\mathcal{A}_i$.

We now explain the inductive equation. Assume that the Y_i-$(X_{i+1} \times \cdots \times X_n)$-tree T_{i+1}^* is the strategy obtained by combining the strategies of processes $P_{i+1}, \ldots, P_n$ and noisy agents $N_{i+2}, \ldots, N_n$. Then the $(X_i; Q_{i+1})$-$(X_{i+1} \times \cdots \times X_n)$-tree $\mathcal{E}_{\lambda_{i+1}}(T_{i+1}^*)$ represents the same strategy but taking into account every possible input over X_i noised by any possible choice of noise specified by the output function λ_{i+1} of $\mathcal{A}_{i+1}$. Consequently, the X_i-$(X_{i+1} \times \ldots \times X_n)$-tree $T'_{i+1} \blacktriangleright \mathcal{E}_{\lambda_{i+1}}(T_{i+1}^*)$ represents the combination of the strategies of processes $P_{i+1}, \ldots, P_n$ and agents $N_{i+1}, \ldots, N_n$. The last operation corresponds to the composition of trees already used by [13] in order to combine the strategies of two processes.

The construction is illustrated in Fig. 9 for a single noisy agent N between two processes S and R, with the architecture of Fig. 8. The expansion of T_R according to λ is an $(X; Q)$-O-tree that gives for any pair consisting of a letter from X and a state of $\mathcal{A}_N$ the output that would be produced by R. Contracting this strategy with respect to T_N is an X-O-tree that gives for any input on X the output of the subcomponent $\xrightarrow{X}\boxed{N, R}\xrightarrow{O}$. Composing with T_S gives a I-$(X \times O)$-tree.

In order to take into account both the input (*i.e.*, the direction) and the label in a tree, [12] associates with a tree T a new tree $xray(T)$, defined by $xray(T)(wa) = (a, T(wa))$ (choosing a dummy direction at the root).

Definition 7. *A distributed strategy $(T_1, \ldots, T_n)$ is called a* winning strategy *for specification φ if* for any *tuple of strategies $(T_{N_2}, \ldots T_{N_n})$ the tree $xray(T_1^*)$ satisfies φ.*

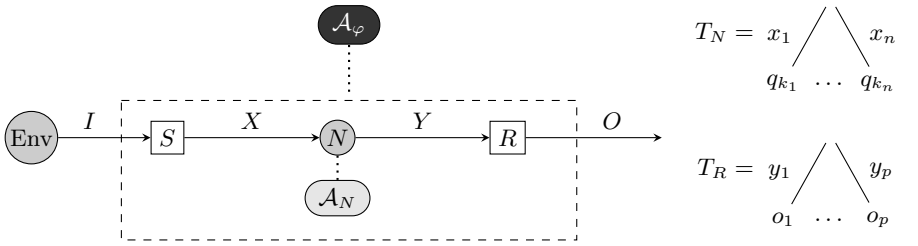


Fig. 8. Noisy architecture for a pair of processes, and strategies T_N, T_R.

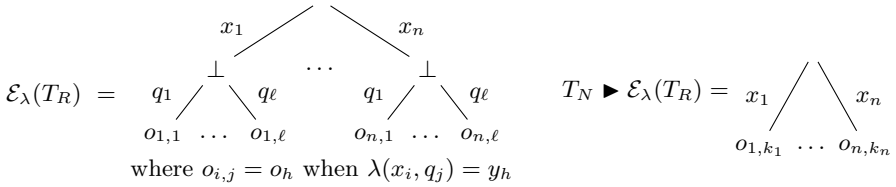


Fig. 9. Composition of strategies.

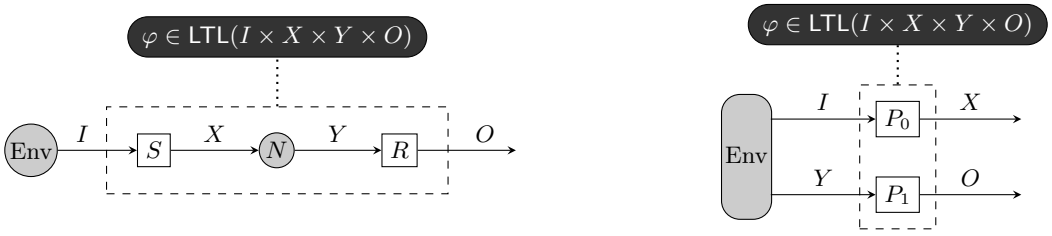
4 The synthesis problem

The synthesis problem for n processes aims at deciding, given a specification φ and n noisy automata, whether there exist winning strategies for the processes.

Example (contd.). Consider the architecture of Fig. 8 where the NA is given in Fig. 7. The goal specification states that the output O exactly matches the input I, that is, in CTL: $\mathsf{AG}\,(I = O)$. Intuitively, it should be clear that the processes S/R cannot have winning strategies, because the input received by R is meaningless due to the power of the noisy agent N.

4.1 Specifying all channels of a noisy pipeline

One could argue that noisy pipelines could be specified by giving a formula constraining the values of all channels, adapting the definition of an outcome to keep in the new $T_1^{*\prime}$ the value of Y_i. However, even in the particular case where $n = 2$, φ is an LTL formula, and N is unconstrained, as Ω in Fig. 10(a), such specification would allow a simulation of the architecture of two processes in parallel, as Θ in Fig. 10(b). The synthesis problem for Θ was proved undecidable by Pnueli & Rosner [17]. The simulation, stated in Theorem 14 (Appendix A), relies on the fact that the information N gets by observing X cannot help the adversaries winning if the processes already have winning strategies.



(a) Noisy architecture Ω specified on all channels.

(b) Architecture Θ, from [17].

Fig. 10. Equivalent undecidable architectures.

4.2 The decidability result for noisy pipelines

We now show the main result of this paper:

Theorem 8. *Given a noisy pipeline with n processes and a* CTL* *specification, the synthesis problem can be decided in k-EXPTIME where $k \sim 2n$.*

Preliminary stage. Before describing the decision procedure, we recall a standard preliminary stage [13] that transforms a CTL* formula φ on $Y_0 \times X_1 \times X_2 \cdots \times X_n$ into an NDTA $\mathcal{A}_\varphi$ on Y_0-$(X_1 \times X_2 \cdots \times X_n)$ trees. The first step is to build an ATA $\mathcal{A}_1$ on Y_0-$(Y_0 \times X_1 \times X_2 \cdots \times X_n)$ trees such that $\mathcal{L}(\mathcal{A}_1)$ is exactly the set of trees satisfying φ. In a second step, $\mathcal{A}_1$ can be translated into an ATA $\mathcal{A}_2$ on Y_0-$(X_1 \times X_2 \cdots \times X_n)$ trees such that a tree T is accepted by $\mathcal{A}_2$ iff $xray(T)$ is accepted by $\mathcal{A}_1$. Finally, the classical construction [15] yields an equivalent NDTA $\mathcal{A}_\varphi$, with a number of states doubly exponential and a number of priorities exponential in the size of φ.

Principle of the decision procedure. The technique used here is based on iterative encapsulation and decomposition. The first problem solved is the case of process P_1, noisy agent N_1 and an abstract process K_1 encapsulating the remaining processes and noisy agents, with a technique described below. Then the resulting automaton which specifies winning strategies for K_1 becomes the new specification to be satisfied on the architecture $\{P_2, N_2, \ldots, N_n, P_n\}$. The last step yields an automaton that specifies the winning strategies for P_n, for which it remains to test the emptiness of its tree language. Thus, the problem reduces to the case where $n = 2$, illustrated in Fig. 8.

Decision procedure for one pair of processes. Given an NDTA $\mathcal{A}_\varphi$ and NA $\mathcal{A}_N$, the goal is to decide whether there exists a winning pair of strategies (T_S, T_R), where strategy T_S of S is an I-X-tree, and strategy T_R for R is a Y-O-tree. More precisely, by a sequence of transformations illustrated in Fig. 11, we build an automaton $\mathcal{A}'_\varphi$ accepting strategies T_R for R such that there exists a strategy T_S for S winning against all strategies T_N of N, as formalized in (∗) (page 3).

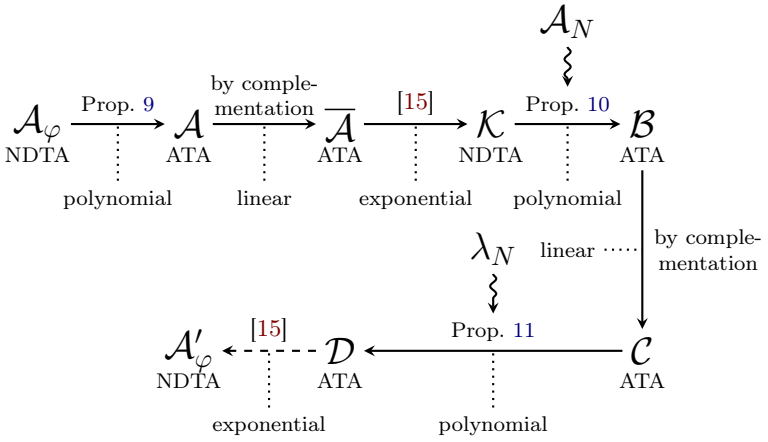


Fig. 11. Sequence of transformations from $\mathcal{A}_\varphi$ to $\mathcal{D}$ or $\mathcal{A}'_\varphi$.

By a construction *à la* Kupferman & Vardi [13], an ATA on X-O-trees $\mathcal{A}$ can be built from $\mathcal{A}_\varphi$ accepting strategies for the abstract process (N, R) such that there exist a strategy for S whose composition satisfies φ. More formally:

Proposition 9. *Let $\mathcal{A}_\varphi$ be a NDTA accepting I-$(X \times O)$ trees. An ATA $\mathcal{A}$ of size polynomial in $|\mathcal{A}_\varphi|$ can be built that verifies: an X-O tree T is accepted by $\mathcal{A}$ if and only if there exists an I-X-tree T' such that the composition $T' \triangleright T$ is accepted by $\mathcal{A}_\varphi$.*

Although this construction, detailed in Appendix B.1, was inspired by the one in [13, Theorem 4.1], it differs slightly since our semantics for composition does not involve a delay. The construction of Proposition 9 corresponds to extracting a tree automaton accepting strategies T to be implemented by the architecture (N, R), i.e. such that there exists a strategy T_S for S and $T_S \triangleright T$ is accepted by $\mathcal{A}_\varphi$. Moreover, $\mathcal{A}$ is an ATA with a polynomial number of states (resp. priorities) in the one of $\mathcal{A}_\varphi$. Therefore an alternating automaton $\overline{\mathcal{A}}$ accepting trees rejected by $\mathcal{A}$, of the same size, can also be built. In order to proceed with the chain (N, R) we transform $\overline{\mathcal{A}}$ into a non deterministic automaton $\mathcal{K}$ at an exponential cost.

In order to obtain an automaton for the strategies of the receiver R, observe that the combination of the strategies T_N (for N) and T_R (for R) is described by the expression $T_N \blacktriangleright \mathcal{E}_{\lambda_N}(T_R)$. Hence:

(1) Given $\mathcal{K}$, we build (in Proposition 10) an alternating automaton $\mathcal{B}$ that accepts trees T' such that *there exists* a strategy T_N accepted by $\mathcal{A}_N$ and $T_N \blacktriangleright T'$ is accepted by $\mathcal{K}$ (otherwise stated, the combined strategy looses).
(2) Complementing $\mathcal{B}$, we obtain an ATA $\mathcal{C}$ accepting trees T' such that *for every* strategy T_N accepted by $\mathcal{A}_N$, the tree $T_N \blacktriangleright T'$ is rejected by $\mathcal{K}$ (otherwise stated, the combined strategy wins).
(3) We build (in Proposition 11) an alternating automaton $\mathcal{D}$ that accepts trees T_R such that $\mathcal{E}_{\lambda_N}(T_R)$ is accepted by $\mathcal{C}$, i.e. such that *for every* strategy T_N accepted by $\mathcal{A}_N$, $T_N \blacktriangleright \mathcal{E}_{\lambda_N}(T_R)$ is rejected by $\mathcal{K}$. Otherwise stated the combined strategy $T_N \blacktriangleright \mathcal{E}_{\lambda_N}(T_R)$ wins.

Iteration and final step. The automaton $\mathcal{D}$ can then be transformed into an NDTA $\mathcal{A}'_\varphi$ so that the sequence above can be reiterated in the general case of n processes (Fig. 6). In the final step, emptiness is tested on the last automaton $\mathcal{D}$ of the iterated sequences.

Proposition 10. *Given an NDTA $\mathcal{K}$ on X-O-trees and an NA $\mathcal{A}_N$ with state set Q and input alphabet X, one can effectively construct an ATA $\mathcal{B}$ such that an $(X;Q)$-O-tree T is accepted by $\mathcal{B}$ if and only if there exists an X-Q-tree T_N which is a strategy for $\mathcal{A}_N$, such that $T_N \blacktriangleright T$ is accepted by $\mathcal{K}$.*

The size of $\mathcal{B}$ is polynomial in that of $\mathcal{K}$ and $\mathcal{A}_N$, and its number of priorities is polynomial in that of $\mathcal{K}$.

The proof can be found in Appendix B.2. Now let ATA $\mathcal{C}$ be the complement of $\mathcal{B}$: it accepts $(X;Q)$-O-trees T such that for any strategy T_N the composition $T_N \blacktriangleright T$ is rejected by $\mathcal{K}$, that is, accepted by $\mathcal{A}$. In order to obtain a Y-O-tree for the strategy of process R, the next step, proved in Appendix B.3, is to transform $\mathcal{C}$ into an ATA $\mathcal{D}$ accepting these trees that, expanded according to the noise function $\lambda = \lambda_N$ of $\mathcal{A}_N$, would be accepted by $\mathcal{C}$.

Proposition 11. *Given an alternating tree automaton $\mathcal{C}$ on two-layer $(X;Q)$-O-trees and a function $\lambda : X \times Q \to Y$, there exists an ATA $\mathcal{D}$ accepting Y-O-trees T such that $\mathcal{E}_\lambda(T)$ is accepted by $\mathcal{C}$. ATA $\mathcal{D}$ has a size and number of priorities polynomial in the ones of $\mathcal{C}$.*

The last step of deciding the emptiness of the language of $\mathcal{D}$ is done in time doubly exponential w.r.t. the size of $\mathcal{D}$ [15,9]. Remark that in the course of the encapsulation procedure described above, R (in this case the encapsulated chain) must be specified by an NDTA $\mathcal{A}'_\varphi$. Hence, ATA

$\mathcal{D}$ is transformed, at exponential cost, into an NDTA $\mathcal{A}'_\varphi$. The complete step of an encapsulation iteration has a doubly exponential blowup: $|\mathcal{A}'_\varphi| = 2^{2^{O(|\mathcal{A}_\varphi|)}}$. Therefore the procedure for n processes is in k-EXPTIME with $k \sim 2n$, hence it is non-elementary.

Example (end). Again for the architecture of Fig. 8 where the NA is given in Fig. 7, applying the sequence of transformations of Fig. 11 (see Appendix C), we obtain an automaton that does not accept any tree, consistently with intuition.

5 Application to covert channel analysis

Covert channels are unforeseen information leakages in systems, exploited by agents who can illegally transfer messages, bypassing the system's security policy. Well-known examples are described in [18,21] for TCP/IP, in which reserved fields of IP packets were used to transmit information.

Covert channel are threats both for security and performance, and several attempts have been made to formalize their existence and detection. The first line of studies, proposed in [1], focused on security policies. Yet, access control does not provide complete solutions for protecting information. Another direction, initiated by [8], consists in representing protocols or systems by models, on which qualitative properties like non-interference [10] or opacity [4] describe the power of observation to discover secret information. However, enforcing these properties overconstrains the system by blocking most communications. Detection and control of covert channels is a weaker requirement, hence a more realistic approach. In previous works, the relationship between input and output in the covert channel is equality, possibly with a bounded delay.

The framework depicted in Fig. 8 (Section 3) represents a typical architecture for a covert channel: processes S and R want to establish a unidirectional communication, from sender S to receiver R, over some protocol that should not allow them to do so. The protocol itself is modeled by the noisy automaton $\mathcal{A}_\mathcal{N}$ for agent N, which can be seen as non-deterministically adding noise to the input X from S. More precisely:

- Information I has to be transmitted by sender S over the covert channel and O corresponds to the interpretation of information Y received by R.
- The noisy agent N represents the operating system or the legal communication protocol and non-determinism is necessary to model the behavior of the other users or internal states of the system unknown to the processes participating in the covert communication.
- Formula φ allows us to express more than simple equality between inputs and outputs. Furthermore, integrating output X of S in the specification φ enables to express requirements about the way the transmission is hidden to the system. The behavior of output Y is not specified in φ since it is already defined by the input/output machine.

In this distributed game, there is a covert channel if and only if there is a winning strategy for S and R satisfying the global specification φ, for any strategy of N following the specification $\mathcal{A}_\mathcal{N}$. The synthesis result thus solves a general covert channel detection problem: if the answer is positive, there is a distributed program satisfying φ, making it possible for S and R to communicate despite N. A negative answer (as in the example of Section 4) shows that no perfect communication can arise. Expressing the specification φ in CTL^* enriches former results where only equality between input and output was considered.

6 Summary and perspectives

We have extended distributed synthesis to a framework in which processes are incomparable with respect to the information they get from the environment, by introducing intermediate adversaries in a pipeline. While specifying on the value of all channels makes the synthesis problem undecidable, describing the adversary power by a nondeterministic finite-state machine makes the synthesis problem decidable, albeit with nonelementary complexity. We showed that this new framework is well suited to model covert channels.

Using the main construction of [13], the decidability proof may be extended to the case where nodes participating to the covert channel also output messages to the environment, that the specification can also constrain. It can also be adapted to the case where we add some fixed delays in the pipeline, to model for instance latency in the network.

However, our decision procedure does not yield winning strategies (when they exist). Computing the strategies in the case of (†) could correspond to building an automaton $\mathcal{A}_\varphi^{T_R}$ such that

$$T_S \text{ is accepted by } \mathcal{A}_\varphi^{T_R} \iff T_S \triangleright T_R \text{ is accepted by } \mathcal{A}_\varphi \qquad (\ddagger)$$

In our case, a similar approach would be to build $\mathcal{A}_\varphi^{T_R}$ such that

$$T_S \text{ accepted by } \mathcal{A}_\varphi^{T_R} \iff \forall T_N,\ T_S \triangleright (T_N \blacktriangleright \mathcal{E}_\lambda(T_R)) \text{ accepted by } \mathcal{A}_\varphi \qquad (**)$$

which amounts to solving a "dual" synthesis problem: building $\mathcal{A}_\varphi^{T_R}$ such that T_S is rejected by $\mathcal{A}_\varphi^{T_R}$ iff there exists $T_{N,R}$ such that $T_S \triangleright T_{N,R}$ is rejected by $\mathcal{A}_\varphi$. This can also be seen as a reverse form of (†), quantifying on the right component instead of the left. Unfortunately the set of such wining strategies *is not necessarily regular* and the existence of automaton $\mathcal{A}_\varphi^{T_R}$ is not ensured. So other techniques should be developed.

Besides, we could add a back arc from the noise N_i to the process P_{i-1}, to model the fact that the latter partially observes *a posteriori* the strategy of the former. It would also be interesting to identify larger classes of architectures where the synthesis problem remains decidable, starting *e.g.*, from uniformly well-connected architectures [7].

References

1. Bell, D.E., Lapadula, L.J.: Secure computer systems: mathematical foundations. Technical Report 2547, MITRE (1973).
2. Berstel, J.: Transductions and Context-Free Languages. BG Teubner (1979).
3. Berwanger, D., Kaiser, Ł.: Information tracking in games on graphs. Journal of Logic, Language and Information (2010).
4. Bryans, J.W., Koutny, M., Mazaré, L., Ryan, P.Y.A.: Opacity generalised to transition systems. International Journal of Information Security **7**(6) (2008) 421–435.
5. Church, A.: Logic, arithmetics, and automata. In: Proc. Int. Congr. Math. (1962) 23–35.
6. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. of LICS'05. (2005) 321–330.
7. Gastin, P., Sznajder, N., Zeitoun, M.: Distributed synthesis for well-connected architectures. In Garg, N., Arun-Kumar, S., eds.: Proc. of FSTTCS'06. Volume 4337 of LNCS, Springer (2006) 321–332.
8. Goguen, J., Meseguer, J.: Security policy and security models. In: Proc.of ieee symposium on security and privacy, IEEE Computer Society Press (1982) 11–20.
9. Grädel, E., Thomas, W., Wilke, Th., eds.: Automata, Logics, and Infinite Games. Number 2500 in LNCS. Springer (October 2002).
10. Hadj-Alouane, N.B., Lafrance, S., Lin, F., Mullins, J., Yeddes, M.: Characterizing intransitive non-interference in security policies with observability. IEEE Trans. on Automatic Control (2004) 920–925.

11. Kupferman, O., Madhusudan, P., Thiagarajan, P., Vardi, M.Y.: Open systems in reactive environments: Control and synthesis. In: Proc. 11th Int. Conf. on Concurrency Theory. Volume 1877 of LNCS, Springer-Verlag (2000) 92–107.
12. Kupferman, O., Vardi, M.Y.: Church's problem revisited. The Bulletin of Symbolic Logic **5**(2) (June 1999) 245–263.
13. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In Halpern, J.Y., ed.: Proc. of LICS'01, Washington, DC, USA, IEEE Computer Society (2001) 389.
14. Madhusudan, P., Thiagarajan, P.S.: Distributed controller synthesis for local specifications. In Orejas, F., Spirakis, P.G., van Leeuwen, J., eds.: Proc. of ICALP'01. Volume 2076 of LNCS, Springer (2001) 396–407.
15. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. Theor. Comput. Sci. **141**(1-2) (1995) 69–107.
16. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. of POPL'89, ACM (1989) 179–190.
17. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. of FOCS'90. Volume II, IEEE Computer Society Press (1990) 746–757.
18. Rowland, C.: Covert Channels in the TCP/IP Protocol Suite. First Monday, Peer reviewed Journal on the Internet (July 1997).
19. Schewe, S., Finkbeiner, B.: Semi-automatic distributed synthesis. International Journal of Foundations Computer Science **18**(1) (2007) 113–138.
20. Thomas, W.: Church's problem and a tour through automata theory. In: Pillars of Computer Science. Volume 4800 of LNCS, Springer (2008) 635–655.
21. Trabelsi, Z., El Sayed, H., Frikha, L., Rabie, T.: A novel covert channel based on the IP header record route option. Int. J. Adv. Media Commun. **1**(4) (2007) 328–350.

A Equivalence of architectures Ω and Θ from Fig. 10

In this section we formalize and prove the claim of Section 4.1.

First the notions of outcome and winning strategy need to be formally defined in this new setting for both games. In the case of Ω, the composition of strategies only involves classical operations on trees, while for Θ, some simple constructions are introduced. We then state and prove the equivalence of the models.

Let π_Z denote the projection of a word on the alphabet Z (seen either as the tuple projection or as the letter-erasure morphism). The set of infinite words over an alphabet Z is denoted by Z^ω. For a (finite or infinite) word $w = z_1 z_2 \dots$ and $0 \leq n \leq |w|$ (the length of w), the word $w_{\leq n} = z_1 \dots z_n$ is the prefix of length n of w.

The superposition of the I-X-tree T_{IX} and the Y-O-tree T_{YO} is the $(I \times Y)$-$(X \times O)$-tree $T_{IX} \oplus T_{YO}$, depicted in Fig. 12 and defined for $w \in (I \times Y)^*$ by:

$$(T_{IX} \oplus T_{YO})(w) = (T_{IX}(\pi_I(w)), T_{YO}(\pi_Y(w))).$$

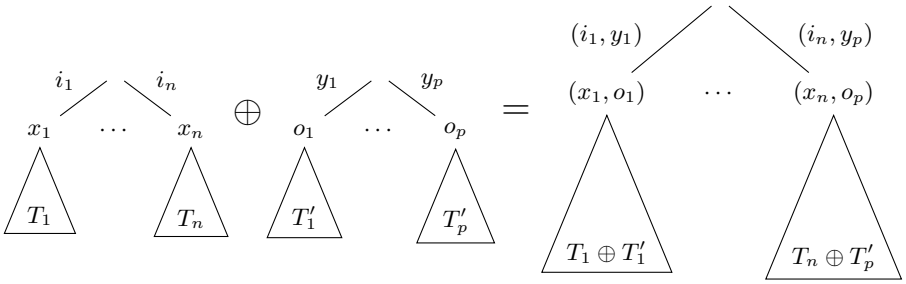


Fig. 12. Superposition of strategies.

We consider in this section an LTL formula φ. Note that although more complicated (*e.g.* CTL*) formulas could be used in the same setting, LTL is sufficient in [17] to prove undecidability of the synthesis problem.

Definition 12. *The semantics of Ω is defined as follows:*

– *Let (T_S, T_R) be a pair of strategies for S and R, respectively, and T_N a strategy for N. The outcome of Ω is*

$$T_\Omega = xray(T_S \triangleright xray(T_N \triangleright xray(T_R))).$$

– *A pair of strategies (T_S, T_R) is winning for specification φ if* for any *strategy T_N, and* any *word $w \in I^\omega$, $T_\Omega(w) \vDash \varphi$.*

Definition 13. *The semantics of Θ is defined as follows:*

– *Let (T_0, T_1) be a pair of strategies for P_0 and P_1, respectively. The outcome of Θ is*

$$T_\Theta = xray(T_0 \oplus T_1).$$

– *A pair of strategies (T_0, T_1) is winning for φ if* for any *word $w \in (I \times Y)^\omega$, $T_\Theta(w) \vDash \varphi$.*

Note that in the above definitions tuples of tuples are assumed to be flattened. In Ω, for a given T_N, the validation of φ for any input from the environment is equivalent to the verification of $T_\Omega \vDash \mathsf{A}\,\varphi$. In Θ, the validation of φ for any input from the environment is equivalent to the verification of $T_\Theta \vDash \mathsf{A}\,\varphi$.

Theorem 14. *Let φ be an* LTL *specification. Processes S and R of architecture Ω have a winning strategy for φ if and only if processes P_0 and P_1 of architecture Θ have a winning strategy for φ.*

Proof. **If Ω has winning strategies.** Suppose (T_S, T_R) is a pair of winning strategies for φ. We shall show that these strategies can be used as strategies for P_0 and P_1, respectively.

Let w be a word of $(I \times Y)^\omega$. Let $u = \pi_I(w)$ (the "I component of w") and $v = \pi_Y(w)$ (the "Y component of w"). Let T_N be the X-Y-tree such that $T_N(t) = v_{|t|}$, *i.e.* the tree whose labels are v_n at depth n. Now consider T_Ω as in Definition 12. Since (T_S, T_R) are winning strategies, $T_\Omega(v) \vDash \varphi$. On the other hand, consider T_Θ as in Definition 13 (with $T_0 = T_S$ and $T_1 = T_R$). For $n \geq 0$, if $w_n = (u_n, v_n)$,

$$\begin{aligned} T_\Omega(u_{\leq n}) &= \Big(u_n, T_S(u_{\leq n}), T_N(\hat{T}_S(u_{\leq n})), T_R(\hat{T}_N(\hat{T}_S(u_{\leq n})))\Big) \\ &= (u_n, T_S(u_{\leq n}), v_n, T_R(v_{\leq n})) \\ T_\Omega(u_{\leq n}) &= (u_n, T_0(u_{\leq n}), v_n, T_1(v_{\leq n})) \\ \text{while} \quad T_\Theta(w_{\leq n}) &= (u_n, v_n, T_0(u_{\leq n}), T_1(v_{\leq n})) \end{aligned}$$

Therefore $T_\Theta(w) = T_\Omega(u)$ (up to a reordering of the 4-uple components), hence $T_\Theta(w) \vDash \varphi$, so (T_0, T_1) is a pair of winning strategies.

If Θ has winning strategies. Suppose (T_0, T_1) is a pair of winning strategies for φ. We shall show that these strategies can be used as strategies for P_S and P_R, respectively.

Let $u \in I^\omega$ be an infinite word and T_N an X-Y-tree (a strategy for N). Let $v = \hat{T}_N(\hat{T}_S(u))$ be the (infinite) word produced by N upon receiving the word produced by S on input u. Let $w = (u, v)$ (that is, for $n \geq 0$, $w_n = (u_n, v_n)$). Now consider T_Θ as in Definition 13. Since (T_0, T_1) are winning strategies, $T_\Theta(w) \vDash \varphi$. On the other hand, consider T_Ω as in Definition 12 (with $T_S = T_0$ and $T_R = T_1$). For $n \geq 0$,

$$\begin{aligned} T_\Theta(w_{\leq n}) &= (u_n, v_n, T_0(u_{\leq n}), T_1(v_{\leq n})) \\ T_\Theta(w_{\leq n}) &= (u_n, T_N(\hat{T}_S(u_{\leq n})), T_S(u_{\leq n}), T_R(\hat{T}_N(\hat{T}_S(u_{\leq n})))) \\ \text{while} \quad T_\Omega(u_{\leq n}) &= \Big(u_n, T_S(u_{\leq n}), T_N(\hat{T}_S(u_{\leq n})), T_R(\hat{T}_N(\hat{T}_S(u_{\leq n})))\Big) \end{aligned}$$

Therefore $T_\Omega(u) = T_\Theta(w)$ (up to a reordering of the 4-uple components), hence $T_\Omega(u) \vDash \varphi$, so (T_S, T_R) is a pair of winning strategies. □

Corollary 15. *The synthesis problem on architecture Ω is undecidable.*

B Proofs for the synthesis decision procedure

B.1 Proof of Proposition 9

Contrary to the composition of [13], there is no initial direction inducing a delay between the action of T' and the ones of T when composing $T' \triangleright T$. Hence, the direction over X has to be remembered in the state of $\mathcal{A}$. More formally, if $\mathcal{A}_\varphi = \langle Q_\varphi, Q^i_\varphi, \delta_\varphi, P_\varphi, \alpha_\varphi \rangle$, then $\mathcal{A} = \langle Q_\mathcal{A}, Q^i_\mathcal{A}, \delta_A, P_\mathcal{A}, \alpha_\mathcal{A} \rangle$ is defined by:

– $Q_{\mathcal{A}} = Q \times X$
– $Q^i_{\mathcal{A}} = Q^i_\varphi \times X$;
– for $r \in Q_\varphi$,

$$\delta_{\mathcal{A}}((r,x),o) = \bigvee_{(r_\eta)_{\eta\in I}\in \delta_\varphi(r,(x,o))} \bigvee_{f:I\to X} \bigwedge_{\eta\in I} ((r_\eta, f(\eta)), f(\eta));$$

– $P_{\mathcal{A}} = P_\varphi$
– $\alpha_{\mathcal{A}}(r,x) = \alpha_\varphi(r)$.

This automaton guesses both a strategy T' on the fly (through a function f) and a choice in the original automaton $\mathcal{A}_\varphi$. If all guesses were correct one a tree T, as enforced by $\mathcal{A}_\varphi$, then the strategy T' that implements the choices for f, when composed with T' is accepted by $\mathcal{A}_\varphi$.

B.2 Proof of Proposition 10

Construction. First, we build $\mathcal{K}$ the NDTA accepting the complement of $\mathcal{L}(\mathcal{A})$. Suppose $\mathcal{K} = \langle K, K^i, \kappa, P_{\mathcal{K}}, \alpha_{\mathcal{K}}\rangle$ and $\mathcal{A}_N = \langle X, Y, Q, q_0, \delta_N, \lambda, F\rangle$, with $P_{\mathcal{K}} = \{0,\dots,p\}$ and, for $k \in K$ and $o \in O$,

$$\kappa(k,o) = \bigvee_{c\in C(k,o)} \bigwedge_{i=1}^{|X|} (k_{c,i}, x_i)$$

where $C(k,o)$ represents the non-deterministic choice of κ. If p is even, then let $P = \{0,\dots,p, p+1\}$; otherwise let $P = \{0,\dots,p,p+1,p+2\}$. In the sequel of the proof, $p_{\max}$ denotes the highest element of P, in this case the odd integer $p+1$ or $p+2$.

Alternating tree automaton $\mathcal{B}$ on $(X;Q)$-O-trees is the tuple

$$\mathcal{B} = \langle (K\times Q\times P)\cup(K\times Q\times P\times X), (K^i\times\{q_0\}\times\{p_{\max}\}), \delta, P, \alpha\rangle$$

where δ is defined by the following equations:

$$\delta((k,q,\ell),o) = \bigvee_{c\in C(k,o)} \bigwedge_{i=1}^{|X|} ((k_{c,i}, q, \ell', x_i), x_i) \quad \text{where } \ell' = \min(\ell, \alpha_{\mathcal{K}}(k_{c,i})) \tag{1}$$

$$\delta((k,q,\ell,x),\bot) = \bigvee_{q'\in\delta_N(q,x)} ((k,q',\ell'),q') \quad \text{where } \ell' = \begin{cases} p_{\max} & \text{if } q\in F \\ \ell & \text{otherwise} \end{cases} \tag{2}$$

Intuitively, on the X-layer, $\mathcal{B}$ acts like $\mathcal{K}$, while on the Q-layer, the choice of the adversary N is simulated. The value ℓ indicates for each state the lowest priority encountered since visiting a state of F. The labeling of the acceptance condition is

$$\alpha(k,q,\ell) = p_{\max} \quad \text{and} \quad \alpha(k,q,\ell,x) = \begin{cases} \ell & \text{if } q\in F \\ p_{\max} & \text{otherwise} \end{cases} \tag{3}$$

Correctness. First remark that acceptance condition α is the conjunction of $\alpha_{\mathcal{K}}$ and F. More precisely, for any infinite sequence ρ of states of $\mathcal{B}$, ρ is accepted with respect to α if and only if the projection of the states onto K is accepted with respect to α_K and in the projection of states onto Q, states of F appear infinitely often. In the following, we will abusively write that states are "in F" if their Q component is, and denote the priority of the K component of a state s simply by "the priority of s with respect to $\alpha_{\mathcal{K}}$". Note that only states of $K \times Q \times P \times X$ matter in the accepting condition. That amounts to considering the acceptance condition only at odd levels of the tree, since (1) ensures that any node at even level will be labeled by a state of priority higher than its children.

Suppose ρ is accepted with respect to α. Then the lowest priority appearing infinitely often, p_0, which is even, cannot be $p_{\max}$, which is odd, hence by (3) F appears infinitely often in ρ. In addition, with infinite occurrences of F, the lowest priority p_0 appearing infinitely often with respect to α is the same as the one with respect to α_K. Indeed, if p_0 appears infinitely often for α, then it appears infinitely often as the lowest priority with respect to $\alpha_{\mathcal{K}}$ between two occurrences of F, therefore appears infinitely often. Any other lower priority $p_1 < p_0$ appearing infinitely often with respect to $\alpha_{\mathcal{K}}$ would also appear as the lowest between two occurrences of F infinitely many times, hence contradicting the minimality of p_0. The converse case follows trivially from the previous observation.

Let T be a tree accepted by $\mathcal{B}$ and ρ be the corresponding run. Let T_N be the X-Q-tree such defined inductively by $T_N(\varepsilon) = q_0$ and for $w = a_1 \cdots a_m \in X^*$, with $\hat{T}_N(w) = q_1 \cdots q_{m-1}$, and for $a_m \in X$, $T(wa_m) = q_m$ where q_m is the only state of Q such that $a_1 q_1 \cdots a_m q_m$ is a branch in ρ. This strategy T_N contains exactly the sequence of choices made by ρ on the transitions of $\mathcal{B}$ defined by (2). An illustration of how this construction works locally is depicted on Fig. 13. By definition of these transitions, the states chosen by T_N always respect the transition relation of $\mathcal{A}_N$. In addition, since every branch of ρ is accepting with respect to α, so is it also on its Q components, which coincide with labels of T_N, with respect to F.

Now consider the contraction $T' = T_N \blacktriangleright T$ of T with respect to N. A run ρ' of T' in $\mathcal{K}$, following the structure of ρ and keeping only the K component of states, can easily be built by (1). A branch in ρ' matches a branch in ρ; since ρ is accepted with respect to α, ρ' is accepted with respect to $\alpha_{\mathcal{K}}$.

Note that if $\mathcal{K}$ had been alternating, different choices could have been made on transitions defined by (2) on different copies. Hence the collection of these choices would not have defined a strategy T_N, where the choice of the next state in $\mathcal{A}_N$ depends only on the history of inputs on X.

Conversely, let T be a tree such that there exists a strategy T_N for $\mathcal{A}_N$ and $\mathcal{K}$ accepts $T' = T_N \blacktriangleright T$. Let ρ' be an accepting run of T' in $\mathcal{K}$. Let ρ be the run of T in $\mathcal{B}$ whose non-deterministic choices are the ones of ρ' on the X level, and the choice of T_N on the Q level. Such a run exists by the definition of δ. Since each branch is accepted by both $\alpha_{\mathcal{K}}$ and F, it is also accepted by α.

B.3 Proof of Proposition 11

Suppose $\mathcal{C} = \langle B, B^i, \beta, P_{\mathcal{B}}, \alpha_{\mathcal{B}} \rangle$ The states of B are separated into B_X and B_Q, and the relation transition κ is separated in two layers:

$$\beta_X(b, o) = \bigvee_{c \in C(b,o)} \bigwedge_{c' \in C'(b,o)} (b_{c,c'}, x_{c,c'}) \qquad (b \in B_X, b_{c,c'} \in B_Q)$$

$$\text{and} \qquad \beta_Q(b, \bot) = \bigvee_{c \in C(b)} \bigwedge_{c' \in C'(b)} (b_{c,c'}, q_{c,c'}) \qquad (b \in B_Q, b_{c,c'} \in B_X)$$

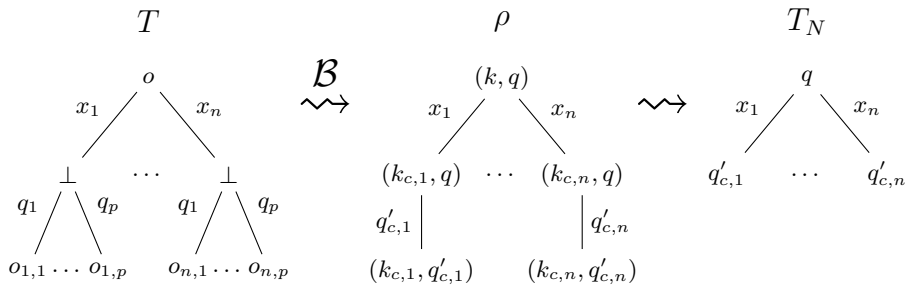


Fig. 13. Obtaining a noise strategy from a run of $\mathcal{B}$. For the sake of readability, irrelevant information (ℓ and x) in states of ρ has been removed.

Moreover, the parity condition is irrelevant on states of B_Q and $B^i \subseteq B_X$, since $(X;Q)$-O-trees start by branching on X. Let $\mathcal{D} = \langle D, D^i, \delta, P_\mathcal{D}, \alpha_\mathcal{D} \rangle$ where $D = B_X$, $D^i = B^i$, $P_\mathcal{D} = P_\mathcal{B}$, $\alpha_\mathcal{D} = \alpha_\mathcal{B}$ and δ is defined by:

$$\delta(d, o) = \bigvee_{c \in C(b,o)} \bigwedge_{c' \in C'(b,o)} \bigvee_{\gamma \in C(b_{c,c'})} \bigwedge_{\gamma' \in C'(b_{c,c'})} (b_{\gamma,\gamma'}, \lambda(x_{c,c'}, q_{\gamma,\gamma'})) \tag{4}$$

One step of transition relation δ takes two steps of ATA $\mathcal{C}$. Two layers of branching are collapsed into one, labeled according to λ. Hence it is straightforward to see that if T is accepted by $\mathcal{D}$, then $\mathcal{E}_\lambda(T)$ is accepted by $\mathcal{C}$.

Conversely, suppose T is such that $T' = \mathcal{E}_\lambda(T)$ is accepted by $\mathcal{C}$. Although T' may contain identical subtrees on which the sub-runs of $\mathcal{C}$ differ, this is captured by the nondeterminism in δ (see equation (4)). The acceptance condition also transfers from a run in $\mathcal{C}$ to a run in $\mathcal{D}$.

C Example for the decision procedure

Consider the architecture of Fig. 8 (with $I = X = Y = O = \{0, 1\}$) with goal specification $\mathsf{AG}\,(I = O)$ and NA $\mathcal{A}_N$ depicted in Fig. 7. More formally, $\mathcal{A}_N$ has states $Q = \{q_i, q_=, q_\neq\}$ and the transition relation δ_N is $\delta_N(q_i, x) = \{q_=, q_\neq\}$, $\delta_N(q_=, x) = \{q_=\}$, and $\delta(q_\neq, x) = \{q_\neq\}$, for $x \in \{0, 1\}$. The output function is $\lambda(x, q_=) = x$ and $\lambda(x, q_\neq) = \overline{x}$, where $\overline{x}$ is the complement of the bit x. The output in q_i is irrelevant in this case.

An NDTA $\mathcal{A}_\varphi$ accepting the I-$(X \times O)$-trees where $I = O$ on all branches can be defined as follows: the state remembers the direction of the branch (hence the input from I) and goes into an error state if the next symbol read does not match the last direction of the branch on its second component. More formally, $\mathcal{A}_\varphi$ has state set $K = \{k_i, k_0, k_1\}$, where k_i is initial (k_i will remain the only initial state throughout this example). The transition relation δ_φ is defined by

$$\delta_\varphi(k_i, (x, \theta)) = \delta_\varphi(k_\theta, (x, \theta)) = (k_0, 0) \wedge (k_1, 1), \quad \text{and} \quad \delta_\varphi(k_\theta, (x, \overline{\theta})) = \mathbf{f}$$

for $x \in X$ and $\theta \in O$. Accepted trees are the ones with only infinite branches (formally, states k_0 and k_1 have priority 0).

First we apply the transformation of Proposition 9 to $\mathcal{A}_\varphi$ in order to obtain a specification for the abstract process (N, R). Automaton $\mathcal{A}$ has states $K \times X$ and transition relation $\delta_\mathcal{A}((k_\theta, x), \overline{\theta}) = \mathbf{f}$

and

$$\begin{aligned}\delta_{\mathcal{A}}((k_i,x),\theta) = \delta_{\mathcal{A}}((k_\theta,x),\theta) = &(((k_0,0),0)\wedge((k_1,1),1))\\ &\vee(((k_0,1),1)\wedge((k_1,0),0))\\ &\vee(((k_0,0),0)\wedge((k_1,0),0))\\ &\vee(((k_0,1),1)\wedge((k_1,1),1))\end{aligned}$$

for $x \in X$ and $\theta \in O$. Again, all states have priority 0. Note that the second component x in the states is actually useless, *i.e.*, replacing the state set $K \times X$ by K and each state (k,x) by k does not change the language accepted by $\mathcal{A}$. (This comes from the fact that X is not used in the specification.) Also remark that the last two conjuncts in the expression of $\delta_{\mathcal{A}}((k_\theta,x),\theta)$ never yield an accepting run. Indeed, using one of them in a transition would require the target node to be labeled by both 0 and 1, in order not to reject the run at one of the children of that node. Therefore, the language accepted by $\mathcal{A}$ is the set of trees such that every node has both children labeled differently.

From this observation, an NDTA $\mathcal{K}$ which accepts the complement of $\mathcal{A}$ can be directly derived, without using the algorithm of [15]: it simply proceeds by guessing a node having both its children labeled by the same bit, as well as this common bit value. The transitions of $\mathcal{K}$ are $\delta_{\mathcal{K}}(k_\theta,\theta) = \mathbf{t}$, $\delta_{\mathcal{K}}(k_\theta,\overline{\theta}) = \mathbf{f}$, and

$$\delta_{\mathcal{K}}(k_i,\theta) = (k_i,0)\vee(k_i,1)\vee((k_0,0)\wedge(k_0,1))\vee((k_1,0)\wedge(k_1,1)) \text{ for } \theta \in O.$$

State k_i is assigned priority 1, hence if it appears infinitely often on a branch, the run is not accepted. Since k_0 and k_1 appear at most once in each branch, their priority is irrelevant and one can let it be also 1. Note that this automaton is already nondeterministic, hence no further transformation step is required.

Now, we shall build automaton $\mathcal{B}$ according to the procedure detailed in the proof of Proposition 10. The set of states of $\mathcal{B}$ is $(K\times Q)\cup(K\times Q\times X)$ (since all states of $\mathcal{K}$ have the same priority 1, it is useless to remember it in the states of $\mathcal{B}$). The transition relation is, for $\theta \in O$, $x \in X$, $q \in Q$, $k \in K$, and $t \in \{=,\neq\}$:

$$\begin{aligned}\delta_{\mathcal{B}}((k_i,q),\theta) &= ((k_i,q,0),0)\ \vee\ ((k_i,q,1),1)\\ &\quad\vee(((k_0,q,0),0)\wedge((k_0,q,1),1))\\ &\quad\vee(((k_1,q,0),0)\wedge((k_1,q,1),1))\\ \delta_{\mathcal{B}}((k_\theta,q),\theta) &= \mathbf{t}\\ \delta_{\mathcal{B}}((k_\theta,q),\overline{\theta}) &= \mathbf{f}\\ \delta_{\mathcal{B}}((k,q_i,x),\bot) &= ((k,q_=),q_=)\vee((k,q_\neq),q_\neq)\\ \delta_{\mathcal{B}}((k,q_t,x),\bot) &= ((k,q_t),q_t).\end{aligned}$$

All states[4] have priority 1, hence infinite branches are rejected.

[4] Formally states paired with q_i should have priority 3, but since all used priorities are odd, choosing 1 instead does not change the accepted language.

Finally, the procedure of the proof of Proposition 11 can be applied to the (syntactic) complement of $\mathcal{B}$, in order to obtain an ATA $\mathcal{D}$ accepting Y-O-trees. The transition relation is

$$\begin{aligned}
\delta_{\mathcal{D}}((k_i, q_i), \theta) = {} & ((k_i, q_=), 1) \wedge ((k_i, q_\neq), 1) \wedge ((k_i, q_=), 0) \wedge ((k_i, q_\neq), 0) \\
& \wedge ((((k_0, q_=), 1) \wedge ((k_0, q_\neq), 0)) \vee (((k_0, q_=), 0) \wedge ((k_0, q_\neq), 1))) \\
& \wedge ((((k_1, q_=), 1) \wedge ((k_1, q_\neq), 0)) \vee (((k_1, q_=), 0) \wedge ((k_1, q_\neq), 1))) \\
\delta_{\mathcal{D}}((k_\theta, q), \theta) = {} & \mathbf{f} \\
\delta_{\mathcal{D}}((k_\theta, q), \overline{\theta}) = {} & \mathbf{t}
\end{aligned}$$

for $\theta \in O$ and $q \in Q$. Now k_i has priority 2, so any infinite branch is accepting.

To show that $\mathcal{D}$ does not accept any tree, consider only the two bottom lines of $\delta_{\mathcal{D}}((k_i, q_i), \theta)$, which is the first transition taken. By distributivity, this yields:

$$\begin{aligned}
& (((k_0, q_=), 1) \wedge ((k_0, q_\neq), 0) \wedge ((k_1, q_=), 1) \wedge ((k_1, q_\neq), 0)) \\
\vee {} & (((k_0, q_=), 1) \wedge ((k_0, q_\neq), 0) \wedge ((k_1, q_=), 0) \wedge ((k_1, q_\neq), 1)) \\
\vee {} & (((k_0, q_=), 0) \wedge ((k_0, q_\neq), 1) \wedge ((k_1, q_=), 1) \wedge ((k_1, q_\neq), 0)) \\
\vee {} & (((k_0, q_=), 0) \wedge ((k_0, q_\neq), 1) \wedge ((k_1, q_=), 0) \wedge ((k_1, q_\neq), 1))
\end{aligned}$$

In any run, each conjunct yields a labeling of each child of the root by both a state corresponding to k_0 and one corresponding to k_1. Hence, whichever symbol will be read then, one of the branches from the root must be non-accepting, due to the transition $\delta_{\mathcal{D}}((k_\theta, q), \theta) = \mathbf{f}$, thus resulting in a non-accepting run. Therefore automaton $\mathcal{D}$ does not accept any tree.

As a result, there are no strategies for S/R to successfully transmit an arbitrary message through the noise adversary of Fig. 7, which indeed conforms to the intuition.