International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Semi-Automatic Distributed Synthesis

SVEN SCHEWE
Universität des Saarlandes, Fachrichtung Informatik, 66123 Saarbrücken, Germany

and

BERND FINKBEINER
Universität des Saarlandes, Fachrichtung Informatik, 66123 Saarbrücken, Germany

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We propose a sound and complete compositional proof rule for distributed synthesis. Applying our proof rule only requires the manual strengthening of the specification into a conjunction of formulas that can be guaranteed by individual black-box processes. All premises of the proof rule can be checked automatically.
For this purpose, we give an automata-theoretic synthesis algorithm for single processes in distributed architectures. The behavior of the local environment of a process is unknown in the process of synthesis and cannot be assumed to be maximal. We therefore consider reactive environments that have the power to disable some of their own actions, and provide methods for synthesis (and realizability checking) in this setting. We establish upper bounds for CTL (2EXPTIME) and CTL* (3EXPTIME) synthesis with incomplete information, matching the known lower bounds for these problems, and provide matching upper and lower bounds for μ-calculus synthesis (2EXPTIME) with complete or incomplete information. Synthesis in reactive environments is harder than synthesis in maximal environments, where CTL, CTL* and μ-calculus synthesis are EXPTIME, 2EXPTIME and EXPTIME complete, respectively.

Keywords: Synthesis, Realizability, Temporal Logic, Concurrency

1. Introduction

In the synthesis of distributed systems, we transform a given specification into a collection of finite-state programs that are guaranteed to satisfy the specification when combined according to a given architecture. For certain restricted classes of architectures, such as pipelines and rings [1, 2], distributed synthesis can be done automatically. However, as soon as the architecture contains an *information fork*, i.e., a pair of processes that each have access to some information about the system state that is hidden from the other process, the problem becomes undecidable [1, 3].

In this paper, we investigate a *semi-automatic* approach, where we synthesize one process at a time. It turns out that the synthesis of a single process can be done automatically and it is always possible to decompose a realizable specification into a conjunction of properties that can be guaranteed by single processes. This approach therefore works for all distributed architectures, including those with information forks.

We consider the problem of synthesizing a single process in a given distributed *architecture*. An architecture consists of an external environment and a set of system processes, which we partition into subsets of *white-box* and *black-box* processes: each white-box process comes with a known and fixed implementation, while the implementation of the black-box processes is to be found. For each process, the architecture identifies a set of input variables and a set of output variables. A process thus does not, in general, have complete information. Based only on the history of previous values of its input variables, the process determines a set of possible assignments to its output variables, among which one particular assignment is then chosen nondeterministically.

The synthesis of process implementations has previously been studied in a simpler setting, where the process is either completely isolated (*closed synthesis*, c.f. [4, 5]) or the process interacts only with a monolithic environment (*open synthesis*). Variations of the open synthesis problem assume the environment to be *maximal*, i.e., to show *all* possible behaviors [6, 7], or to be *reactive*, i.e., to show some but not necessarily all of its possible behaviors [8].

The main difference between the classic synthesis problems and the problem studied in this paper is that the environment of the process now consists of multiple constituents. In addition to the external environment, the process may interact with the white-box processes and with the other black-box processes in the system. Both the behavior of the external environment and the behavior of the white-box processes are known *a priori*: In our setting, we assume that the behavior of the external environment is maximal and that the behavior of each white-box process is given as a (possibly nondeterministic) finite-state automaton. By contrast, the strategies of the other black-box processes are unknown. From the point of view of the considered process, their behavior therefore appears reactive: at any point, they may disable some (but not all) of their possible responses.

We call a process implementation *resilient* if the specification is satisfied *independently* of how the other black-box processes are implemented. We demonstrate that the resilient synthesis problem is 2EXPTIME-complete for CTL and μ-calculus specifications and 3EXPTIME complete for specifications in CTL*. Our proof is constructive: we give an automata-theoretic algorithm that determines for a temporal specification and a process in a distributed architecture whether there exists a resilient implementation and, if yes, computes one such implementation. We establish 2EXPTIME and 3EXPTIME upper bounds for synthesis with incomplete information in case of μ-calculus and CTL* specifications, respectively. These upper bounds match the lower bounds for checking resilient realizability for CTL and CTL* specifications, respectively, under the assumption of complete information and a monolithic environment [8].

We propose to use the new synthesis algorithm in a compositional synthesis rule for distributed synthesis. Applying our synthesis rule requires the strengthening of the specification into a conjunction of local specifications for the individual processes, such that each local specification is resiliently realized by its process. The rule is complete: if a specification can be implemented, then there also exists a strengthening for which that implementation is resilient. Since the synthesis of resilient implementations is automatic, the strengthening is the only manual step in the application of the rule.

The remainder of the paper is structured as follows. In Section 2, we formally introduce the synthesis problem studied in this paper. We introduce the compositional synthesis rule in Section 3 and illustrate its application on a simple example in Section 4. In Section 5, we prove the completeness of the rule. The synthesis algorithm is presented in Section 6.

2. The Distributed Synthesis Problem

In the *distributed synthesis* problem, we construct for a given pair (A, φ), consisting of an architecture A and a specification φ, a finite-state program (or *strategy*) for each black-box process in A, such that the joint behavior satisfies φ.

2.1. Architectures

An architecture is a tuple $A = (B, W, \mathit{env}, V, \mathcal{I}, \mathcal{O}, \{s_w \mid w \in W\})$ consisting of:

- A set B of black-box processes for which an implementation is sought.
- A set W of white-box processes for which an implementation is provided.
- A designated environment-process *env*.
  The disjoint union $P = B \uplus W \uplus \{\mathit{env}\}$ of the black- and white-box processes and the environment is called the set of processes.
- A set V of shared boolean variables, which the processes use to communicate. They also serve as atomic propositions in the specification.
- A collection $\mathcal{I} = \{I_p \subseteq V \mid p \in P\}$, consisting of the input variables for the processes.
- A collection $\mathcal{O} = \{O_p \subset V \mid p \in P\}$, consisting of the output variables for the processes. The elements of $\mathcal{O}$ partition the set V of system variables, i.e., each variable is written to by exactly one process. We assume that O_p is non-empty for all black- and white-box processes.
- A set $\{s_w : (2^{I_w})^* \to 2^{2^{O_w}} \smallsetminus \{\emptyset\} \mid w \in W\}$ of finite-state implementations for the white-box processes.

The environment is a special process, which is always omniscent ($I_{env} = V$) and shows maximal behavior, i.e., $s_{env} : (2^V)^* \to \{2^{O_{env}}\}$.

2.2. Specifications

We consider specifications that are given as a CTL, CTL*, or μ-calculus formula φ. (Appendix A contains a brief summary of these logics.) The models of such a specification φ are a set $\mathcal{M}_\varphi$ of total 2^{AP}-labeled Υ-trees over some set Υ of directions, where $AP = V$ denotes the set of atomic propositions in φ.

As usual, an Υ-*tree* is given as a prefix-closed subset $Y \subseteq \Upsilon^*$ of all finite words over Υ. If the set of directions is not important or clear from the context, we call Y a tree. We define that every non-empty node $x \cdot \upsilon$, $x \in \Upsilon^*, \upsilon \in \Upsilon$, has the direction $dir(x \cdot \upsilon) = \upsilon$ and the empty word ε has some designated *root-direction* $dir(\varepsilon) = \upsilon_0 \in \Upsilon$. An Υ-tree Y is called *total*, if it contains the empty word $\varepsilon \in Y$ and every element $y \in Y$ of the tree has at least one successor $y \cdot \upsilon \in Y, \upsilon \in \Upsilon$. If $Y = \Upsilon^*$, the tree is called *full*.

For given finite sets Σ and Υ, a Σ-*labeled* Υ-*tree* is a pair $\langle Y, l\rangle$, consisting of a tree $Y \subseteq \Upsilon^*$ and a labeling function $l : Y \to \Sigma$ that maps every node of Y to a letter of Σ.

As usual, we write $\langle Y, l\rangle \vDash \varphi$ for a tree $\langle Y, l\rangle \in \mathcal{M}_\varphi$ which is a model of φ.

2.3. Implementations

We consider nondeterministic implementations. After reading the values of its input variables, each process determines a set of possible assignments to its output variables, among which one particular assignment is then chosen nondeterministically. We require that at least one future must be possible; i.e., the set of assignments cannot be empty. The set of possible decisions $\mathfrak{D}_p \equiv 2^{2^{O_p}} \smallsetminus \{\emptyset\}$ therefore form the set of nonempty sets of assignments to the output variables of p[a]. Since the sets of output variables O_p and O_q of two different processes p and q are disjoint, their sets $\mathfrak{D}_p$ and $\mathfrak{D}_q$ are disjoint, too.

A process p is implemented by a *strategy*, i.e., a function $s_p : (2^{I_p})^* \to \mathfrak{D}_p \equiv 2^{2^{O_p}} \smallsetminus \{\emptyset\}$. s_p maps each input history of p to a non-empty set of possible variable assignments. A strategy is *finite-state* (or regular) if it is the unraveling of a finite-state transducer. We consider only finite-state implementations in this paper, and denote finite-state strategies by $s_p : (2^{I_p})^* \to_r \mathfrak{D}_p$.

The implementations $\{s_w : (2^{I_w})^* \to_r \mathfrak{D}_w \mid w \in W\}$ of the white-box processes W are fixed for the architecture. An *implementation* of an architecture is a set of strategies $S = \{s_b : (2^{I_b})^* \to_r \mathfrak{D}_b \mid b \in B\}$ for the black-box processes.

We identify a strategy $s_p : (2^{I_p})^* \to \mathfrak{D}_p$ with the full $\mathfrak{D}_p$-labeled 2^{I_p}-tree $\langle (2^{I_p})^*, s_p\rangle$.

[a]For generality, we allow all processes to be nondeterministic. If a subset $S \subseteq B \uplus W$ of the processes is to be deterministic, one can simply choose, for all $p \in S$, the set $\mathfrak{D}_p$ of possible decisions of p as the set of singleton subsets of 2^{O_p} instead of the set of non-empty subsets. Generally, $\mathfrak{D}_p$ can be set to any non-empty subset of $2^{2^{O_p}} \smallsetminus \{\emptyset\}$.

2.4. Compositions

The strategies of the processes are composed synchronously. In every step, each process $p \in P$ fixes, based on the history visible to p, a set $d \in \mathfrak{D}_p$ of possible valuations of its output variables O_p. The composition of a set $Q \subseteq P$ of processes maps the global input history to the possible valuation of their joint output variables $\bigcup_{p \in Q} O_p$, which is defined by the decisions of the strategies $\{s_q \mid q \in Q\}$ of the processes in Q.

As an auxiliary notion, we first define the composition of sets:

- For two sets of sets A and B, we define their composition $A \oplus B = \{a \cup b \mid a \in A, b \in B\}$ as the set of unions of their elements.
- For two sets of sets of sets A and B, we define their composition $A \otimes B = \{a \oplus b \mid a \in A, b \in B\}$ as the set of compositions of their elements.

We use $\mathfrak{D}_Q$ to abbreviate the set $\bigotimes_{q \in Q} \mathfrak{D}_q$ of possible joint decisions of the processes in Q. The *composition* of two strategies s_p and s_q of processes p and q *with complete information* ($I_p = I_q = V$) is the strategy $s_{\{p,q\}} : (2^V)^* \to \mathfrak{D}_p \otimes \mathfrak{D}_q$ with $s_{\{p,q\}}(y) = s_p(y) \oplus s_q(y)$ for all $y \in (2^V)^*$.

In the general case of incomplete information we need to compose strategies of processes with different sets of input variables. We define two auxiliary functions: The function *hide* projects a history of variable assignments to a history for a subset of the variables (such as the visible input variables of a process). The function *widen* extends a strategy to a larger set of input variables, without changing its behavior (i.e., the extended strategy does not depend on the new input variables).

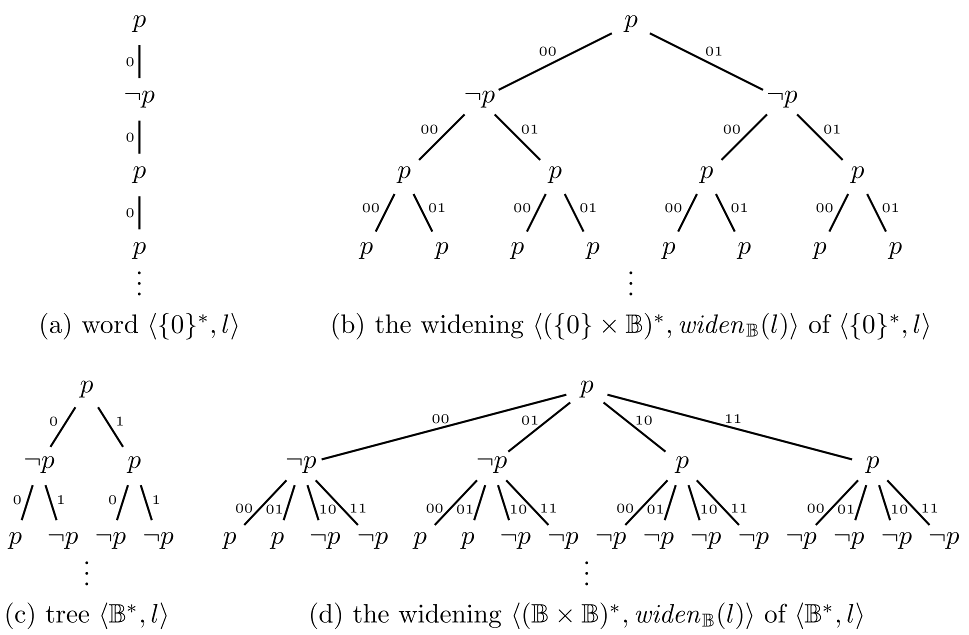
- For a set $\Xi \times \Upsilon$ of directions and a node $x \in (\Xi \times \Upsilon)^*$, $\mathit{hide}_\Upsilon(x)$ denotes the node in Ξ^* obtained from x by replacing (ξ, υ) by ξ in each letter of x.
- For a Σ-labeled Ξ-tree $\langle \Xi^*, l \rangle$, $\langle (\Xi \times \Upsilon)^*, \mathit{widen}_\Upsilon(l) \rangle$, denotes the Σ-labeled $\Xi \times \Upsilon$-tree $\langle (\Xi \times \Upsilon)^*, l' \rangle$ with $l'(x) = l(\mathit{hide}_\Upsilon(x))$. Hence, $l \circ \mathit{hide}_\Upsilon \equiv \mathit{widen}_\Upsilon(l)$.

The widening function widen_Υ guarantees that the label of each node $y \in (\Xi \times \Upsilon)^*$ in the resulting tree depends only on the visible part $\mathit{hide}_\Upsilon(y) \in \Xi^*$ of the input history. This construction is illustrated with an example in Figure 1.

We define the *composition* $s_{\{p,q\}} : (2^V)^* \to \mathfrak{D}_{\{p,q\}} = \mathfrak{D}_p \otimes \mathfrak{D}_q$ of two strategies $s_p : (2^{I_p})^* \to \mathfrak{D}_p$ and $s_q : (2^{I_q})^* \to \mathfrak{D}_q$ as the strategy $s_{\{p,q\}} = \mathit{widen}_{2^{V \smallsetminus I_p}}(s_p) \oplus \mathit{widen}_{2^{V \smallsetminus I_q}}(s_q)$, and use s_Q as an abbreviation for the composition $\bigotimes_{q \in Q} s_q$ of the strategies $\{s_q \mid q \in Q\}$ of the processes in Q.

2.5. Computations

An implementation $\{s_b : (2^{I_b})^* \to \mathfrak{D}_b \mid b \in B\}$ defines the *computation tree* $\langle \|s_P\|, \mathit{dir} \rangle$, where $s_P = \bigotimes_{p \in P} s_p$ denotes the composition of the process strategies, and $\|s_P\|$ denotes the set of computations allowed by s_P: $\|s_P\|$ is the greatest total tree $Y \subseteq (2^V)^*$ such that for all $y \in (2^V)^*$ and all $\upsilon \in 2^V$, if $y \cdot \upsilon \in Y$, then $y \in Y$ and $\upsilon \in s_P(y)$.



(a) word $\langle \{0\}^*, l\rangle$ (b) the widening $\langle (\{0\}\times\mathbb{B})^*, \mathit{widen}_{\mathbb{B}}(l)\rangle$ of $\langle \{0\}^*, l\rangle$

(c) tree $\langle \mathbb{B}^*, l\rangle$ (d) the widening $\langle (\mathbb{B}\times\mathbb{B})^*, \mathit{widen}_{\mathbb{B}}(l)\rangle$ of $\langle \mathbb{B}^*, l\rangle$

Figure 1: The widening function widen_Υ maps a Ξ-tree $\langle \Xi^*, l\rangle$ to the $\Xi\times\Upsilon$-tree whose label depends only on the Ξ-part of the history. Figure 1a shows a unary tree (or word) as a simple input to the widening function $\mathit{widen}_{\mathbb{B}}$. The result is a binary tree, where every path is labeled identically (Figure 1b). Figure 1d shows the result of a Boolean widening on the Boolean tree $\langle \mathbb{B}, l\rangle$ from Figure 1c. Here, every pair $y, y' \in (\mathbb{B}\times\mathbb{B})^*$ of nodes which are indistinguishable under hiding of the second element ($\mathit{hide}_{\mathbb{B}}(y) = \mathit{hide}_{\mathbb{B}}(y')$) has the same label $l(\mathit{hide}_{\mathbb{B}}(y))$.

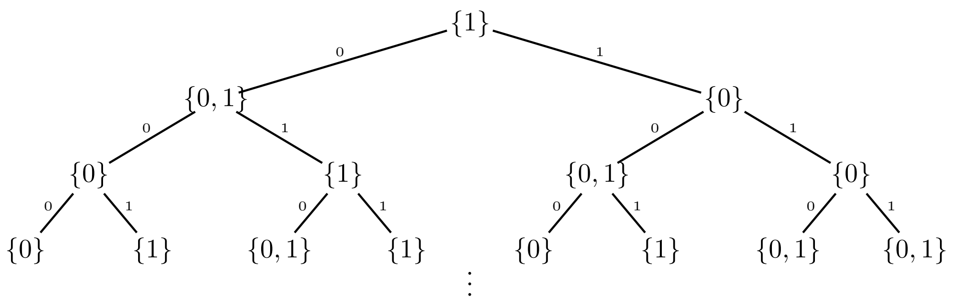
Figure 2 illustrates the construction of the computation tree with an example. A system with composed strategy s_P satisfies a specification φ if and only if the computation tree $\langle \|s_P\|, \mathit{dir}\rangle \vDash \varphi$ is a model of φ.

2.6. Realizability

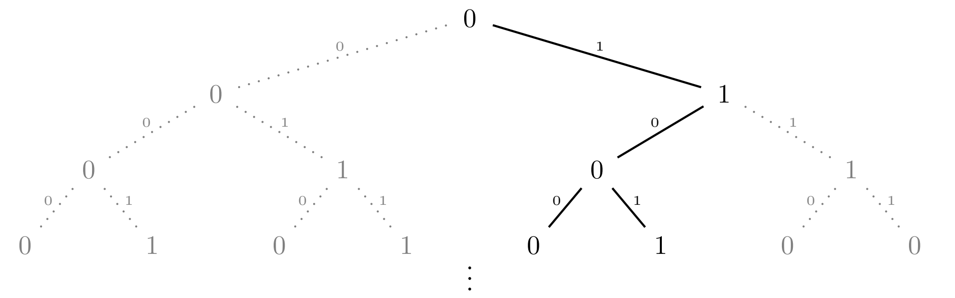
A set of strategies $\{s_q : (2^{I_q})^* \to \mathfrak{D}_q \mid q \in Q\}$ of a set $Q \subseteq B$ of black-box processes is called a *resilient realization* of a specification φ if the computation tree satisfies φ independently of the other black-box processes: i.e., for every $\mathfrak{D}_{B\smallsetminus Q}$-labeled 2^V-tree $\langle (2^V)^*, s_{B\smallsetminus Q}\rangle$, representing the behavior of the black-box processes in $B\smallsetminus Q$, the computation tree $\langle \|s\|, \mathit{dir}\rangle$ with $s = s_Q \otimes s_{B\smallsetminus Q} \otimes s_W \otimes s_{env}$ is a model of φ.

A specification φ is *resiliently realizable* by a set $Q \subseteq B$ of black-box processes, denoted by $(A, Q) \overset{\exists}{\vDash} \varphi$, if there exists a set of finite-state strategies[b] for the processes

[b]For existential and universal quantification over a set of strategies for a set of processes, we use the notation $\exists\{s_p : 2^{I_p} \to \mathfrak{D}_p \mid p \in Q\}$ and $\forall\{s_p : 2^{I_p} \to \mathfrak{D}_p \mid p \in Q\}$, respectively.



(a) strategy tree $\langle \mathbb{B}^*, s_P : \mathbb{B}^* \to 2^{2^{\mathbb{B}}} \setminus \{\emptyset\}\rangle$



(b) computation tree $\langle \| s_P \|, \mathit{dir}\rangle$

Figure 2: Figure 2a shows a Boolean strategy tree $\langle \mathbb{B}^*, s_P\rangle$. Every node is labeled with a nonempty subset of $\mathbb{B}$, indicating the possible futures. Figure 2b shows (in solid lines) the computation tree $\langle \| s_P \|, \mathit{dir}\rangle$. The computation tree contains those paths of the full tree $\langle \mathbb{B}, \mathit{dir}\rangle$ (shown in gray) that are consistent with the strategy.

in Q that are a resilient realization of φ:

$$\begin{aligned}(A, Q) \vDash \varphi :\Leftrightarrow \exists \{s_b : (2^{I_b})^* \to_r \mathfrak{D}_b \mid b \in Q\}. \forall \{s_{B \setminus Q} : (2^V)^* \to \mathfrak{D}_{B \setminus Q}.\\ \langle \| \bigotimes_{b \in Q} s_b \otimes s_{B \setminus Q} \otimes s_W \otimes s_{env} \|, \mathit{dir}\rangle \vDash \varphi\end{aligned}$$

A specification φ is *realizable* if it is (resiliently) realizable by the entire set B of black-box processes.

3. The Compositional Synthesis Rule

Using the definitions from the previous section, we now introduce a semi-automatic approach to distributed synthesis. We define a compositional synthesis rule that establishes the realizability of a specification by showing that the specification can be strengthened into a conjunction of local specifications for the individual processes, such that each local specification is resiliently realized by its process. While the strengthening of the specification must be done manually, we will show in Section 6 that all premises of the synthesis rule can be checked automatically.

For a distributed architecture A with a set of black-box processes $B = \{b_1, \cdots, b_n\}$, and CTL* or μ-calculus formulas $\psi, \varphi_{b_1}, \dots \varphi_{b_n}$,

$$\begin{array}{cccl}
\text{(R1)} & (A, \{b_1\}) & {}^{\exists}\!\models & \varphi_{b_1} \\
\vdots & & \vdots & \\
(\mathrm{R}n) & (A, \{b_n\}) & {}^{\exists}\!\models & \varphi_{b_n} \\
\text{(S)} & (A, \emptyset) & {}^{\exists}\!\models & \bigwedge_{b \in B} \varphi_b \rightarrow \psi \\
\hline
 & (A, B) & {}^{\exists}\!\models & \psi
\end{array}$$

Theorem 1 *The compositional synthesis rule is sound.*

Proof. Premises (R1) through (Rn) prove that each local specification φ_{b_i} is resiliently realized by the respective black-box process b_i:

$$(A, \{b_i\})^{\exists}\!\models \varphi_{b_i} \Leftrightarrow \exists s_{b_i} : (2^{I_{b_i}})^* \rightarrow_r \mathfrak{D}_{b_i}. \forall s_{B \smallsetminus \{b_i\}} : (2^V)^* \rightarrow \mathfrak{D}_{B \smallsetminus \{b_i\}}. \\ \langle \| s_{b_i} \otimes s_{B \smallsetminus \{b_i\}} \otimes s_W \otimes s_{env} \|, dir \rangle \models \varphi_{b_i}.$$

Consequently, such strategies s_{b_i} can be fixed independently. The resulting implementation $\{s_{b_i} \mid b_i \in B\}$ satisfies φ_{b_i} for all $b_i \in B$. Hence, $(A, B)^{\exists}\!\models \bigwedge_{b_i \in B} \varphi_{b_i}$ holds true:

$$\exists \{s_{b_i} : (2^{I_{b_i}})^* \rightarrow_r \mathfrak{D}_{b_i} \mid b_i \in B\}. \langle \| \bigotimes_{b_i \in B} s_{b_i} \otimes s_W \otimes s_{env} \|, dir \rangle \models \bigwedge_{b_i \in B} \varphi_{b_i}.$$

Premise (S) shows that the conjunction of the local specifications $\bigwedge_{b_i \in B} \varphi_{b_i}$ implies the system specification ψ. Therefore, every implementation that satisfies all local specifications must also satisfy ψ:

$$(A, \emptyset)^{\exists}\!\models \bigwedge_{b_i \in B} \varphi_{b_i} \rightarrow \psi$$
$$\Leftrightarrow \forall s_B : (2^V)^* \rightarrow \mathfrak{D}_B. \langle \| s_B \otimes s_W \otimes s_{env} \|, dir \rangle \models \bigwedge_{b_i \in B} \varphi_{b_i} \rightarrow \psi$$
$$\Leftrightarrow \forall s_B : (2^V)^* \rightarrow \mathfrak{D}_B. \langle \| s_B \otimes s_W \otimes s_{env} \|, dir \rangle \models \bigwedge_{b_i \in B} \varphi_{b_i} \Rightarrow \langle \| s_B \otimes s_W \otimes s_{env} \|, dir \rangle \models \psi.$$

In particular, the computation tree defined by $\{s_{b_i} \mid b_i \in B\}$ must satisfy ψ. Hence, the compositional synthesis rule is sound. $\square$

The compositional proof rule can be applied to all architectures, including those which contain an information fork. This is no contradiction to the undecidability of the realizability problem for these architectures: there is no decision procedure which can distinguish unrealizable specifications from realizable specifications for which we merely failed to find local strengthenings.

4. Example

We illustrate the compositional synthesis rule with a simple distributed *shared-resource* application. The system architecture is shown in Figure 3a. The external environment Env is depicted as a circle, the two black-box processes p_1 and p_2 as filled rectangles, and the white-box "Arbiter" process as a white rectangle.

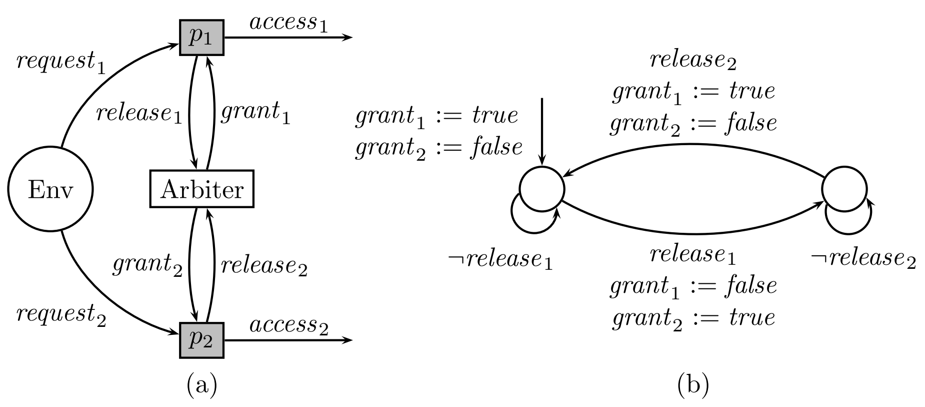


Figure 3: A simple distributed shared-resource application. (a) The system architecture. An edge between two process nodes p and q labeled with variable v indicates that v is an output variable of process p and an input variable of process q. (b) The implementation of the white-box process Arbiter, represented as a finite-state Mealy machine. The edges are labeled with the value of the input variables and with new assignments to the output variables (if the value of the output variables changes).

Env can request *access* to the resource by setting the *request* variable of one of the two black-box processes p_1 and p_2. The white-box process Arbiter, whose implementation is shown in Figure 3b, ensures mutual exclusion by passing a *grant* back and forth between p_1 and p_2, such that each process retains the grant until the respective *release* variable is set.

We specify the expected behavior of the shared-resource system as a conjunction $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3$ of three CTL* formulas, where the first two formulas specify that there is a way for both processes to use the resource infinitely often ($\psi_i =$ EGF $access_i$ for $i \in \{1, 2\}$) and the third formula specifies mutual exclusion ($\psi_3 =$ AG $\neg(access_1 \wedge access_2)$).

Obviously, neither p_1 nor p_2 can guarantee ψ for *all* possible implementations of the other process (for example, if p_1 constantly sets its $access_1$ variable to *true*, p_2 cannot avoid violating mutual exclusion if it is to obtain access along some branch).

Using the proof rule, we need to strengthen ψ into two separate properties φ_{p_1} and φ_{p_2} that can be resiliently realized by p_1 and p_2, respectively. A natural assumption to be made by process p_{3-i} about process p_i is that there is a path such that process p_i infinitely often releases the grant ($\alpha_1^{p_i} =$ EGF $release_i$) and that, on every path, p_i only accesses the resource when permitted by the arbiter ($\alpha_2^{p_i} =$ AG $access_i \to grant_i$). By adding these assumptions, we obtain a strengthened specification $\varphi = \varphi_{p_1} \wedge \varphi_{p_2}$ where

$$\varphi_{p_i} \;=\; \alpha_1^{p_i} \wedge \alpha_2^{p_i} \quad \wedge \quad (\alpha_1^{p_{3-i}} \wedge \alpha_2^{p_{3-i}} \;\to\; \psi).$$

Once the auxiliary formulas φ_{p_1} and φ_{p_2} have been defined, a resilient implementation can be found for both processes. For example, process p_i can guarantee

φ_{p_i} by setting $access_i$ after each $request_i$ as soon as $grant_i$ becomes *true* and by setting $release_i$ in the immediately following state.

In terms of the traditional system development process, the strengthening in our proof rule can be understood as the definition of an abstract interface or contract between the processes. Typically, the user can choose from multiple correct contracts. In the shared-resource application, the user might, for example, alternatively specify strict turn-taking by strengthening ψ to φ'_{p_1} and φ'_{p_2}, where φ'_{p_1} requires that p_1 accesses the resource exactly after every *odd* number of steps and φ'_{p_2} requires that p_2 accesses the resource exactly after every *even* number of steps. The resilient implementations of this strengthening are slightly unorthodox (the arbiter is ignored), but are also guaranteed to satisfy ψ.

As a final remark about this example, let us convince ourselves that it is really necessary to consider *resilient* implementations. Suppose that, in a hypothetical alternative proof rule, we require only the (cheaper) realizability in a maximal environment. It is now possible to strengthen ψ into two formulas φ''_{p_1} and φ''_{p_2} that can, in a maximal environment, be guaranteed by p_1 and p_2, respectively, but whose conjunction is equivalent to *false*. For example, p_1 can easily guarantee $AG\,\neg release_1$, while any implementation of p_2 guarantees $EF release_1$ in a maximal environment.

5. Completeness

To demonstrate the completeness of the compositional synthesis rule, we show that the auxiliary formulas $\varphi_{b_1}, \ldots, \varphi_{b_n}$ required in the rule can be derived from a distributed implementation $\{s_{b_i} \mid b_i \in B\}$ that satisfies the specification ψ. Given an implementation s_b of a black-box process b, we define a CTL formula φ_b that is a *strict characterization* of the behavior of b and the white-box processes. Strict characterization means that (1) s_b is a resilient implementation of φ_b and (2) for all other implementations $\{s'_{b_i} \mid b_i \in B\}$ that realize φ_b, the implementations s_b and s'_b have the same computation trees:

$$\forall s_{B\smallsetminus\{b\}} : (2^V)^* \to \mathfrak{D}_{B\smallsetminus\{b\}}.\ \|s_b \otimes s_{B\smallsetminus\{b\}} \otimes s_W \otimes s_{env}\| = \|s'_b \otimes s_{B\smallsetminus\{b\}} \otimes s_W \otimes s_{env}\|.$$

Condition (1) guarantees that premise (Ri) of the proof rule is satisfied for each black-box process b_i, and Condition (2) guarantees that premise (S) is satisfied.

The arbiter depicted in Figure 3(b) has the following strict characterization:

$$\varphi_{arbiter} = \varphi_0 \wedge \mathrm{AG}(\varphi_0 \vee \varphi_1), \text{ with}$$

$$\begin{aligned}\varphi_0 = {} & grant_1 \wedge \neg grant_2 \wedge \mathrm{AX}(release_1 \to \neg grant_1 \wedge grant_2)\\ & \wedge \mathrm{AX}(\neg release_1 \to grant_1 \wedge \neg grant_2), \text{ and}\end{aligned}$$

$$\begin{aligned}\varphi_1 = {} & \neg grant_1 \wedge grant_2 \wedge \mathrm{AX}(release_2 \to grant_1 \wedge \neg grant_2)\\ & \wedge \mathrm{AX}(\neg release_2 \to \neg grant_1 \wedge grant_2).\end{aligned}$$

Let the implementation for the processes $b \in B$ and for the white-box processes be given as finite-state transducers, which we combine into a single transducer T_B

with a set S_B of states and an initial state $s_0^B \in S_B$. Additionally, we construct a finite-state transducer T_b with a set S_b of states and initial state s_0^b for the product of each single black-box process b and the white-box processes.

We define a CTL formula φ_b such that the models of φ_b are the trees obtained by unraveling T_b. To construct φ_b, we give a formula φ_s for each state $s \in S_b$ which ensures that, for the next $max\{|S_B|, |S_b|\} + 1$ steps, the tree corresponds to the unraveling of T starting in state s. Since the other black-box processes are unknown, φ_s does not require that all branches of the unraveling exist, but rather that, provided they do exist, the reaction is in accordance with T_b. The specification $\varphi_b = \varphi_{s_0^b} \wedge \mathrm{AG} \bigvee_{s \in S_b} \varphi_s$ requires that φ_{s_0} holds initially, and that always some φ_s holds true.

The formula φ_b is a strict characterization of the behavior of b and the white-box processes. As required by Condition (1), s_b is a resilient implementation of φ_b. For Condition (2), note that resilient realization includes realization in a maximal environment as a special case: hence, $s'_b \otimes \bigotimes_{w \in W} s_w$ must react to any input from the other black-box processes or from the external environment exactly like $s_b \otimes \bigotimes_{w \in W} s_w$.

An unraveling of height $|S_b| + 1$ suffices for strict characterizations, and an unraveling of height $|S_B| + 1$ suffices to guarantee that the formula $\bigwedge_{b \in B} \varphi_B$ is a strict characterization of the overall system.

Theorem 2 *The compositional synthesis rule is complete.*

Proof. Assume that ψ is realizable and let $\{s_{b_i} : (2^{I_{b_i}})^* \rightarrow_r \mathfrak{D}_{b_i} \mid b_i \in B\}$ be a realization of ψ. For each black-box process b_i, we can infer a formula φ_{b_i}, which is a strict characterization of the behavior of b_i and the white-box processes, i.e., of $s_{b_i} \otimes s_W$. Premises (Ri) of the compositional synthesis rule are satisfied because for each φ_{b_i}, the given s_{b_i} is a resilient realization.

The conjunction of the single strict characterizations define a strict specification of the overall system. Consequently, each implementation that satisfies $\bigwedge_{b_i \in B} \varphi_{b_i}$ also satisfies ψ, and Premise (S) holds true. $\square$

6. Synthesis of Resilient Implementations

We now develop a procedure that checks if a specification is resiliently realizable by a single black-process b, $(A, \{b\})^{\exists}\!\models \varphi$, as required for Premises (R1) through (Rn), and a procedure that checks if a specification is resiliently realized by the empty set of black-box processes, $(A, \emptyset)^{\exists}\!\models \varphi$, as required for Premise (S).

Every formula of a temporal logic can be translated into an alternating tree automaton that accepts exactly its set of models. This automaton is the starting point for our construction, which consists of a series of tree automata transformations.

6.1. Tree Automata

An *alternating parity tree automaton* is a tuple $\mathcal{A} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$, where Q denotes a finite set of states, $q_0 \in Q$ denotes a designated initial state, δ denotes a transition function, and $\alpha : Q \to C \subset \mathbb{N}$ is a coloring function. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon)$ maps a state and an input letter to a positive boolean combination of states and directions (for a predefined finite set Υ of directions).

An alternating automaton runs on full Σ-labeled Υ-trees. A *run tree* on a given full Σ-labeled Υ-tree $\langle \Upsilon^*, l \rangle$ is a $Q \times \Upsilon^*$-labeled tree where the root is labeled with (q_0, ε) and where for every node n with a label (q, y) and a set $child(n)$ of children, the labels of these children have the following properties:

- for all children $m \in child(n)$ of n, the label of m is $(q_m, y \cdot \upsilon_m)$ for some $q_m \in Q$ and $\upsilon_m \in \Upsilon$ such that (q_m, υ_m) is an atom of $\delta(q, l(y))$, and
- the set of atoms defined by the children of n satisfies $\delta(q, l(y))$[c].

The label of a node on the run tree refers to the current state of the automaton and its current position on the input tree. An infinite path fulfills the *parity condition*, if the highest color of the states appearing infinitely often on the path is even. A run tree is *accepting* if all infinite paths fulfill the parity condition. Note, that a run tree may have *finite* paths: if $\delta(q, l(y))$ is *true*, then the set of successors may be empty. Likewise, an input tree may not have any run tree. If $\delta(q, l(y))$ is *false*, then no set of successors can satisfy $\delta(q, l(y))$. Finite paths are always accepting[d].

A total Σ-labeled Υ-tree is accepted if it has an accepting run tree.

The set of trees accepted by an alternating automaton $\mathcal{A}$ is called its *language* $\mathcal{L}(\mathcal{A})$. $\overline{\mathcal{L}(\mathcal{A})}$ denotes the set of full Σ-labeled Υ-trees not accepted by $\mathcal{A}$. An automaton is empty if its language is empty.

The acceptance of a tree can also be viewed as the outcome of a game, where player *accept* chooses, for a pair $(q, \sigma) \in Q \times \Sigma$, a set of atoms of $\delta(q, \sigma)$, satisfying $\delta(q, \sigma)$, and player *reject* chooses one of these atoms, which is executed. The input tree is accepted iff player *accept* has a strategy enforcing a path that fulfills the parity condition. One of the player has a memoryless winning strategy, i.e., a strategy where the moves only depend on the state of the automaton, the position in the tree and, for player *reject*, on the choice of player *accept* in the same move.

A *nondeterministic* automaton is a special alternating automaton, where the image of δ consists only of such formulae that, when rewritten in disjunctive normal form, contain exactly one element of $Q \times \{\upsilon\}$ for all $\upsilon \in \Upsilon$ in every disjunct.

For nondeterministic automata, every node of a run tree corresponds to a node in the input tree. Emptiness can therefore be checked with an *emptiness game*, where player *accept* also chooses the letter of the input alphabet. A nondeterministic automaton is empty iff the emptiness game is won by *reject*.

Symmetric alternating automata are a variant of alternating automata that run on total Σ-labeled Υ-trees. For a symmetric alternating automaton $\mathcal{S} =$

[c]If $\delta(q, l(y))$ is rewritten in disjunctive normal form, we can require that the atoms of a disjunct are contained in the set of atoms defined by the children of n.

[d]If only infinite paths shall be allowed, *true* and *false* can be substituted by fresh states q_{true} and q_{false}, respectively, such that

- $\alpha(q_{true})$ is even and $\alpha(q_{false})$ is odd, and
- for some $\upsilon \in \Upsilon$ and all $\sigma \in \Sigma$, $\delta(q_{true}, \sigma) = (q_{true}, \upsilon)$ and $\delta(q_{false}, \sigma) = (q_{false}, \upsilon)$.

$(\Sigma, Q, q_0, \delta, \alpha)$, Q, q_0, and α are defined as before. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \{\Box, \Diamond\})$ now maps a state and an input letter to a positive boolean combination over atoms that refer to *some* ($\Diamond$) or *all* ($\Box$) successor states.

A *run tree* on a given Σ-labeled Υ-tree $\langle R, r\rangle$ is a $Q \times \Upsilon^*$-labeled tree where the root is labeled with (q_0, ε) and where, for a node n with a label (q, y) and a set of labels of its successors $L = \{r(n \cdot \rho) \mid \rho \in \mathit{succset}(n)\}$, the following property holds: there is a set of atoms $A \subseteq 2^{Q \times \{\Box,\Diamond\}}$ satisfying $\delta(q, l(y))$ such that $\forall q' \in Q.((q', \Box) \in A \Rightarrow \forall \upsilon \in \mathit{succset}(x).(q', y \cdot \upsilon) \in L) \wedge ((q', \Diamond) \in A \Rightarrow \exists \upsilon \in \mathit{succset}(x).(q', y \cdot \upsilon) \in L)$. A *run tree* on a given Σ-labeled Υ-tree $\langle \Upsilon^*, l\rangle$ is a $Q \times \Upsilon^*$-labeled tree where the root is labeled with (q_0, ε) and where for every node n with a label (q, y) and a set $\mathit{child}(n)$ of children, the labels of these children have the following properties:

- for all children $m \in \mathit{child}(n)$ of n, the label of m is $(q_m, y \cdot \upsilon_m)$ for some $q_m \in Q$ and $\upsilon_m \in \Upsilon, y \cdot \upsilon_m \in Y$ such that $(q_m, \Box)$ or $(q_m, \Diamond)$ is an atom of $\delta(q, l(y))$, and
- interpreting each occurrence of $(q', \Box)$ as $\bigwedge_{\upsilon \in \Upsilon.\, y\cdot\upsilon \in Y}(q', \upsilon)$ and each occurrence of $(q', \Diamond)$ as $\bigvee_{\upsilon \in \Upsilon.\, y\cdot\upsilon \in Y}(q', \upsilon)$, the atoms defined by the children of n satisfy $\delta(q, l(y))$.

We introduce a function $\mathit{suc} : (Q \times \Sigma \to \mathbb{B}^+(Q \times \{\Box, \Diamond\})) \to (Q \times \Sigma \times 2^\Upsilon \smallsetminus \{\emptyset\} \to \mathbb{B}^+(Q \times \Upsilon))$ that translates the transition function of a symmetric alternating automaton running on total Σ-labeled Υ-trees into the corresponding transition function of an alternating automaton running on full $\Sigma \times 2^\Upsilon \smallsetminus \{\emptyset\}$-labeled Υ-trees. For the set $2^\Upsilon \smallsetminus \{\emptyset\}$ of possible sets of successors, $\mathit{suc}(\delta) : Q \times \Sigma \times (2^\Upsilon \smallsetminus \{\emptyset\}) \to \mathbb{B}^+(Q \times \Upsilon)$ maps a state, an input letter and a set of successors to a positive boolean combination of states and directions.

6.2. Overview

To check whether a specification φ is resiliently realizable by a black-box process b, we construct a tree automaton $\mathcal{C}_\varphi$ that accepts the set of resilient realizations, and check $\mathcal{C}_\varphi$ for emptiness. Our construction consists of a series of automata transformations. Before we describe the individual steps in detail we give an overview of the construction.

From formulas to automata. We start our construction with a symmetric automaton $\mathcal{S}_\varphi$ that accepts the models of the specification φ:

$$\mathcal{L}(\mathcal{S}_\varphi) = \{\langle Y, l : Y \to 2^V\rangle \mid \langle Y, l\rangle \vDash \varphi\}.$$

Characteristic trees. Computation trees are total trees. Since it is more convenient to work with automata over full trees, we make the choice of enabled directions explicit. We add the set of enabled directions to the label and construct an alternating automaton $\mathcal{A}_\varphi$ from $\mathcal{S}_\varphi$ that uses this information: Where $\mathcal{S}_\varphi$ sends a copy to all successors, $\mathcal{A}_\varphi$ sends a copy to all *enabled* successors, and where $\mathcal{S}_\varphi$ sends a copy to some successor, $\mathcal{A}_\varphi$ sends a copy to some enabled successor.

Considering a full $2^V \times (2^{2^V} \smallsetminus \{\emptyset\})$-labeled 2^V-tree $\langle (2^V)^*, l' \rangle$, where the nodes are additionally decorated with the sets of relevant successors, one can easily determine the original total 2^V-labeled 2^V-tree $\langle Y, l \rangle$, which we call its *characteristic tree*. We continue with an automaton $\mathcal{A}_\varphi^+$ that accepts those full $2^V \times (2^{2^V} \smallsetminus \{\emptyset\})$-labeled 2^V-trees whose characteristic tree is a model of φ. (Note that the labeling of nodes that do not belong to the characteristic tree have no influence on the acceptance.) Each distributed implementation defines a set of successors, i.e., a tree $\langle (2^V)^* \to \bigotimes_{p\in P} s_p \rangle$, whose label refers to the set of successors. Since the mapping of $\bigotimes_{p\in P} s_p$ is resticted to $\mathfrak{D}_b \otimes \mathfrak{D}_W \otimes \mathfrak{D}_{B\smallsetminus\{b\}} \otimes \mathfrak{D}_{env}$ we restrict the language of $\mathcal{A}_\varphi^+$ to $2^V \times \mathfrak{D}_b \otimes \mathfrak{D}_W \otimes \mathfrak{D}_{B\smallsetminus\{b\}} \otimes \mathfrak{D}_{env}$-labeled 2^V-trees, resulting in an automaton $\mathcal{A}_\varphi$ with

$$\begin{aligned}\mathcal{L}(\mathcal{A}_\varphi) = \{\langle (2^V)^*, l \times l' \rangle \mid\ & l : (2^V)^* \to 2^V, \\ & l' : (2^V)^* \to \mathfrak{D}_b \otimes \mathfrak{D}_W \otimes \mathfrak{D}_{B\smallsetminus\{b\}} \otimes \mathfrak{D}_{env} \text{ and } \langle \|l'\|, l \rangle \in \mathcal{L}(\mathcal{S}_\varphi)\}.\end{aligned}$$

Resilience. In this step, we universally quantify over the decisions of the remaining black-box processes $B \smallsetminus \{b\}$ and the environment. We build an automaton $\mathcal{R}_\varphi$ that accepts a $2^V \times \mathfrak{D}_b \otimes \mathfrak{D}_W$-labeled 2^V-tree if all $\mathfrak{D}_{B\smallsetminus\{b\}} \otimes \mathfrak{D}_{env}$ extensions are accepted by $\mathcal{A}_\varphi$:

$$\begin{aligned}\mathcal{L}(\mathcal{R}_\varphi) = \{\langle (2^V)^*, l \times l' \rangle \mid\ & l : (2^V)^* \to 2^V,\ l' : (2^V)^* \to \mathfrak{D}_b \otimes \mathfrak{D}_W \text{ and} \\ & \forall l'' : (2^V)^* \to \mathfrak{D}_{B\smallsetminus\{b\}} \otimes \mathfrak{D}_{env}.\ \langle (2^V)^*, l \times (l' \otimes l'') \rangle \in \mathcal{L}(\mathcal{A}_\varphi)\}.\end{aligned}$$

Pruning directions from the labeling. The valuation of the atomic propositions $V = AP$ must for each node be consistent with its direction. The alternating automaton $\mathcal{D}_\varphi$ accepts a $\mathfrak{D}_b \otimes \mathfrak{D}_W$-labeled 2^V-tree if the $2^V \times \mathfrak{D}_b \otimes \mathfrak{D}_W$-labeled 2^V-tree obtained by adding the direction of a node to the label is accepted by $\mathcal{R}_\varphi$:

$$\mathcal{L}(\mathcal{D}_\varphi) = \{\langle (2^V)^*, l : (2^V)^* \to \mathfrak{D}_b \otimes \mathfrak{D}_W \rangle \mid \langle (2^V)^*, dir \times l) \rangle \in \mathcal{L}(\mathcal{R}_\varphi)\}.$$

Adjusting for white-box processes. The behavior of the white-box processes is known and fixed. In this step we build an automaton $\mathcal{W}_\varphi$ that simulates their behavior and supplies their decisions. The alternating automaton $\mathcal{W}_\varphi$ accepts a $\mathfrak{D}_b$-labeled 2^V-tree if the $\mathfrak{D}_b \otimes \mathfrak{D}_W$-labeled 2^V-tree obtained by adding the decisions of the white-box processes is accepted by $\mathcal{D}_\varphi$:

$$\mathcal{L}(\mathcal{W}_\varphi) = \{\langle (2^V)^*, l : (2^V)^* \to \mathfrak{D}_b \rangle \mid \langle (2^V)^*, l \otimes s_W) \rangle \in \mathcal{L}(\mathcal{D}_\varphi)\}.$$

Localization. The alternating automaton $\mathcal{B}_\varphi$ accepts a $\mathfrak{D}_b$-labeled 2^{I_b}-tree if its proper widening is accepted by $\mathcal{W}_\varphi$:

$$\mathcal{L}(\mathcal{B}_\varphi) = \{\langle (2^{I_b})^*, l : (2^{I_b})^* \to \mathfrak{D}_b \rangle \mid \langle (2^V)^*, widen_{2^{V\smallsetminus I_p}}(l) \rangle \in \mathcal{L}(\mathcal{W}_\varphi)\}.$$

Emptiness check. Checking $\mathcal{B}_\varphi$ for emptiness decides realizability: $(A,\{b\})^{\exists}\models\varphi$ holds true iff $\mathcal{B}_\varphi$ is non-empty. To check emptiness, we first transform $\mathcal{B}_\varphi$ into an equivalent nondeterministic automaton $\mathcal{C}_\varphi$, and then check the emptiness of $\mathcal{C}_\varphi$ by solving its emptiness game. A memoryless winning strategy in the emptiness game defines a finite-state implementation $s_b : (2^{I_b})^* \rightarrow_r \mathfrak{D}_p$ for the process b that resiliently realizes φ.

In the following, we discuss the automata transformations in detail.

6.3. Automata Transformations

6.3.1. From Formulas to Automata

We use standard constructions to translate a temporal specification φ into a symmetric alternating automaton $\mathcal{S}_\varphi$ that accepts the models of the formula.

Theorem 3 *Given a CTL specification φ, we can construct a symmetric alternating automaton $\mathcal{S}_\varphi$ with $O(|\varphi|)$ states and two colors such that the language of $\mathcal{S}_\varphi$ consists of the models of φ [9]. Given a CTL* specification φ, we can construct a symmetric alternating automaton $\mathcal{S}_\varphi$ with $2^{O(|\varphi|)}$ states and five colors such that the language of $\mathcal{S}_\varphi$ consists of the models of φ [9]. Given a μ-calculus specification φ, we can construct a symmetric alternating automaton $\mathcal{S}_\varphi$ with $O(|\varphi|^2)$ states and $O(|\varphi|)$ colors such that the language of $\mathcal{S}_\varphi$ consists of the models of φ [7].* $\square$

6.3.2. Characteristic Trees

Computation trees are total trees. Automata transformations are simpler for automata running on full trees; we therefore represent total trees as full trees by decorating each node with its own set of successors (making the choice of enabled directions explicit). When the set of enabled directions is added to the label, we can construct an alternating automaton $\mathcal{A}_\varphi$ from $\mathcal{S}_\varphi$ that uses this information: Where $\mathcal{S}_\varphi$ sends a copy to all successors, $\mathcal{A}_\varphi$ sends a copy to all *enabled* successors, and where $\mathcal{S}_\varphi$ sends a copy to some successor, $\mathcal{A}_\varphi$ sends a copy to some enabled successor.

For a $\Sigma\times\Xi$-labeled Υ-tree $\langle Y,l\rangle$, we denote the Σ-*projection* $proj_\Sigma : \langle Y,l\rangle \mapsto \langle Y,l_\Sigma\rangle$ with $l(y)=(\sigma,\xi)\Rightarrow l_\Sigma : y\mapsto\sigma$ that maps $\Sigma\times\Xi$-labeled Υ-trees to Σ-labeled Υ-trees.

For a full $\Sigma\times(2^\Upsilon\smallsetminus\{\emptyset\})$-labeled Υ-tree $\langle\Upsilon^*,l\rangle$, we define the *characteristic tree* as the total Σ-labeled Υ-tree $\langle Y,l_c\rangle = char(\langle\Upsilon^*,l\rangle)$ to be the sub-tree of $proj_\Sigma(\langle\Upsilon^*,l\rangle)$ with $Y=\|l_{suc}\|$ for $\langle\Upsilon^*,l_{suc}\rangle = proj_{2^\Upsilon\smallsetminus\{\emptyset\}}(\langle\Upsilon^*,l\rangle)$. Intuitively, the second argument in the label of $\langle\Upsilon^*,l\rangle$ defines the set of successors of a node.

Lemma 1 *Given a symmetric alternating automaton $\mathcal{S}=(\Sigma,Q,q_0,\delta,\alpha)$, running on total Σ-labeled Υ-trees, we can construct an alternating automaton $\mathcal{A}=(\Sigma\times(2^\Upsilon\smallsetminus\{\emptyset\}),\Upsilon,Q,q_0,suc(\delta),\alpha)$ that accepts a full $\Sigma\times(2^\Upsilon\smallsetminus\{\emptyset\})$ labeled Υ-tree $\langle\Upsilon^*,l\rangle$, iff $char(\langle\Upsilon^*,l\rangle)$ is accepted by $\mathcal{S}$.*

Proof. Let $\langle T, l_T \rangle = \mathit{char}(\langle \Upsilon^*, l \rangle)$. Then the successor set $\mathit{succset}(x)$ of a node $x \in T$ is defined by the label of x for the tree $\langle \Upsilon^*, l \rangle$: $\mathit{succset}(x) = \mathit{proj}_{2^\Upsilon \smallsetminus \{\emptyset\}}(l(x))$. $\square$

6.3.3. Resilience

In this step we establish independence from the choices of the remaining black-box processes by constructing an alternating automaton $\mathcal{R}_\varphi$ that accepts a $2^V \times \mathfrak{D}_b \times \mathfrak{D}_W$-labeled 2^V-tree if all $\mathfrak{D}_{B \smallsetminus \{b\}} \times \mathfrak{D}_{env}$ extensions are accepted by $\mathcal{A}_\varphi$. Intuitively, the automaton guesses the most hostile behavior of the remaining black-box processes. Note that for *universal* specifications the most hostile behavior of the remaining black-box processes is the *maximal* behavior, where each process $p \in B \smallsetminus \{b\}$ continuously enables all successors ($s_p(y) = \{2^{O_p}\}\ \forall y \in (2^V)^*$). The quantification step can therefore be avoided in case of universal specifications (see Subsection 6.6 for details).

To construct $\mathcal{R}_\varphi$, we interpose a language projection between two language complementations:

(i) We complement $\mathcal{A}_\varphi$, i.e., we compute an alternating automaton $\mathcal{I}_\varphi$ with $\mathcal{L}(\mathcal{I}_\varphi) = \overline{\mathcal{L}(\mathcal{A}_\varphi)}$.
(ii) Next, we build a nondeterministic automaton $\mathcal{N}_\varphi$ with the same language $\mathcal{L}(\mathcal{N}_\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$.
(iii) Then, we compute a nondeterministic automaton $\mathcal{P}_\varphi$ that accepts a $2^V \times \mathfrak{D}_b \times \mathfrak{D}_W$-labeled 2^V-tree if it is the the $\mathfrak{D}_{B \smallsetminus \{b\}} \times \mathfrak{D}_{env}$-projection of a tree accepted by $\mathcal{N}_\varphi$.
(iv) Finally, we complement $\mathcal{P}_\varphi$, i.e., we compute an alternating automaton $\mathcal{R}_\varphi$ with $\mathcal{L}(\mathcal{R}_\varphi) = \overline{\mathcal{L}(\mathcal{P}_\varphi)}$.

An alternating automaton $\mathcal{A} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$ can be complemented by dualizing its transition function (i.e., replacing each occurrence of $\wedge$, $\vee$, *true* and *false* by $\vee$, $\wedge$, *false* and *true*, respectively) and increasing the color of each state by one. Dualization of alternating automata was introduced by Muller and Schupp [10].

Lemma 2 *[10] Given an alternating automaton $\mathcal{A} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$, the dual automaton $\mathcal{I} = (\Sigma, \Upsilon, Q, q_0, \overline{\delta}, \alpha + 1)$, where $\overline{\delta}$ is the function dual to δ, accepts a tree $\langle \Upsilon^*, l \rangle$ iff $\langle \Upsilon^*, l \rangle$ is not accepted by $\mathcal{A}$.* $\square$

The most expensive operation in our construction is the transformation of general alternating tree automata to nondeterministic automata.

Lemma 3 *[3, 11] Given an alternating automaton $\mathcal{A}$ with n states and c colors, we can construct an equivalent nondeterministic automaton $\mathcal{N}$ with $n^{O(c \cdot n)}$ states and $O(c \cdot n)$ colors.* $\square$

Language projection is a standard operation on nondeterministic automata, where the automaton nondeterministically chooses a suitable extension of the label of an input tree.

Lemma 4 *Given a nondeterministic automaton $\mathcal{N} = (\Sigma \times \Xi, \Upsilon, Q, q_0, \delta, \alpha)$, we can construct a nondeterministic automaton $\mathcal{P} = (\Sigma, \Upsilon, Q, q_0, \delta', \alpha)$ that accepts a Σ-labeled Υ-tree $\langle \Upsilon^*, l \rangle$ iff there is a $\Sigma \times \Xi$-labeled Υ-tree $\langle \Upsilon^*, l_\Xi \rangle$ accepted by $\mathcal{N}$ with $\langle \Upsilon^*, l \rangle = proj_\Sigma(\langle \Upsilon^*, l_\Xi \rangle)$.*

Proof. $\mathcal{P}$ can be constructed by using δ' to guess the correct tree: we set $\delta' : (q, \sigma) \mapsto \bigvee_{\xi \in \Xi} \delta(q, (\sigma, \xi))$. $\square$

6.3.4. Pruning Directions from the Labeling

We are only interested in those trees where the label of every node is in accordance with its direction. This information then becomes redundant and can be pruned. The following operation performs this pruning; the state space of the resulting automaton is linear in the state space of the original automaton, while the set of colors remains unchanged.

For a Σ-labeled Υ-tree $\langle \Upsilon^*, l \rangle$, we define the function $xray : \langle \Upsilon^*, l \rangle \mapsto \langle \Upsilon^*, l' \rangle$ with $l'(x) = (dir(x), l(x))$ that maps Σ-labeled Υ-trees to $\Upsilon \times \Sigma$-labeled Υ-trees.

Lemma 5 *[9] Given an alternating automaton $\mathcal{R} = (\Upsilon \times \Sigma, \Upsilon, Q, q_0, \delta, \alpha)$, we can construct an alternating automaton $\mathcal{D} = (\Sigma, \Upsilon, Q \times \Upsilon, (q_0, \upsilon_0), \delta', \alpha')$ that accepts $\langle \Upsilon^*, l \rangle$ iff $\mathcal{R}$ accepts $xray(\langle \Upsilon^*, l \rangle)$.* $\square$

The transition function $\delta' : Q \times \Upsilon \times \Sigma \to \mathcal{B}^+(Q \times \Upsilon \times \Upsilon)$ can be constructed from $\delta : Q \times \Upsilon \times \Sigma \to \mathcal{B}^+(Q \times \Upsilon)$ by replacing all occurrences of (q, υ) in each $\delta(q', \upsilon', \sigma')$ by (q, υ, υ), storing the direction as quasi-input. $\alpha' : (q, c) \mapsto \alpha(q)$ simply evaluates the first component of the new states.

6.3.5. Adjusting for White-Box Processes

The $\mathfrak{D}_W$-fraction of the label represents the decisions made by the white-box processes. Consequently, we are only interested in those trees, where the label of every node is in accordance with these decisions. This information is then redundant and can be pruned. We assume that the composed strategy $\bigotimes_{w \in W} s_w$ of the white-box processes is represented as a finite-state automaton $\mathcal{O} = (2^V, O, o_0, d_W, o_W)$, where O is a set of states, o_0 the initial state, the transition function $d_W : 2^V \times O \to O$ is a mapping from the input alphabet and the set of states to the set of states, and the output function $o_W : O \to 2^{2^{O_W}} \smallsetminus \{\emptyset\}$ maps each state to a nonempty set of output letters. The following operation performs the pruning; the state space of the resulting automaton is linear in the state space of the original automaton and the number of states of $\mathcal{O}$, while the set of colors remains unchanged.

Lemma 6 *Given an alternating automaton $\mathcal{D} = (\Sigma \times \Xi, \Upsilon, Q, q_0, \delta, \alpha)$ and a finite automaton $\mathcal{O} = (\Sigma, O, o_0, d_W, o_W)$ that produces a Ξ-labeled Υ-tree $\langle \Upsilon^*, l \rangle$, we can construct an alternating automaton $\mathcal{W} = (\Sigma, \Upsilon, Q \times O, (q_0, o_0), \delta', \alpha')$ that accepts $\langle \Upsilon^*, l' \rangle$ iff $\mathcal{D}$ accepts $\langle \Upsilon^*, l'' \rangle$ with $l'' : y \mapsto (l'(y), l(y))$.*

Proof. If $\delta : (q, \sigma, \xi) \mapsto b_{(q,\sigma,\xi)}(\{q_i, \upsilon_i \mid i \in I\})$, we can set $\delta' : (q, o, \sigma) \mapsto b_{(q,\sigma,o_W(o))}(\{q_i, d_W(\sigma, o), \upsilon_i \mid i \in I\})$. The coloring function can simply be set to $\alpha' : (q, o) \mapsto \alpha(q)$. $\square$

6.3.6. Localization

The process b is in general not omniscient, and its output may only depend on the history of the input visible to b. The following transformation therefore accepts a 2^{O_p}-labeled 2^{I_p}-tree if its proper widening is accepted by $\mathcal{W}_\varphi$. The state space and the set of colors remain unchanged.

Lemma 7 *[9] Given an alternating automaton $\mathcal{W} = (\Sigma, \Xi \times \Upsilon, Q, q_0, \delta, \alpha)$, we can construct an alternating automaton $\mathcal{B} = (\Sigma, \Xi, Q, q_0, \delta', \alpha)$ that accepts a tree $\langle \Xi^*, l\rangle$ iff $\mathcal{W}$ accepts $\langle (\Xi \times \Upsilon)^*, widen_\Upsilon(l)\rangle$.* □

The narrowing operation transforms the transition function δ. The new transition function δ' is constructed by replacing each occurrence of $(q, (\xi, \upsilon))$ in the mapping of δ by (q, ξ). A memoryless winning strategy of $\mathcal{B}$ for a Σ-labeled Ξ-tree $\langle \Xi^*, l\rangle$ defines a memoryless winning strategy for $\mathcal{W}$ on its Υ-widening $widen_\Upsilon(\langle \Xi^*, l\rangle)$: if the winning strategy of $\mathcal{B}$ maps a node $x \in \Xi^*$ and a state $q \in Q$ to a set $\{(q_i, \xi_i) \mid i \in I\}$ of atoms, then the winning strategy of $\mathcal{D}$ maps a state $y \in (\Xi \times \Upsilon)^*$ with $hide_\Upsilon(y) = x$ and q to $\{(q_i, (\xi_i, \upsilon)) \mid i \in I, \upsilon \in \Upsilon\}$.

Vice versa, an accepting run-tree of $\mathcal{D}$ for $widen_\Upsilon(\langle \Xi^*, l\rangle)$ can be turned into an accepting run tree of $\mathcal{B}$ for a Σ-labeled Ξ-tree $\langle \Xi^*, l\rangle$ by replacing every node $y \in (\Xi \times \Upsilon)^*$ from each label of the run tree by $hide_\Upsilon(y)$.

6.3.7. Emptiness Check

We check the emptiness of $\mathcal{B}_\varphi$ by first constructing a nondeterministic automaton $\mathcal{C}_\varphi$ and then testing the emptiness of $\mathcal{C}_\varphi$ (Lemma 3).

Language emptiness for a nondeterministic automaton $\mathcal{C}_\varphi$ can be checked by solving its emptiness game. If the language is non-empty, a winning memoryless strategy can be used to construct a finite-state transducer, which generates a regular tree in the language of $\mathcal{C}_\varphi$.

Lemma 8 *Given a nondeterministic automaton $\mathcal{C} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$ with n states and c colors, we can check language emptiness in time $n^{O(c)}$. If $\mathcal{C}$ is non-empty, we can construct a regular tree accepted by $\mathcal{C}$ within the same complexity bound.*

Proof. The emptiness game can be modeled as a two person parity game with $n^{O(1)}$ states[e] and c colors. Such a game can be solved in $n^{O(c)}$ time [12]. A winning strategy defines a transducer with at most n states, which represents a regular tree accepted by $\mathcal{C}$. □

6.4. Upper Bounds

Our construction establishes 2EXPTIME upper bounds for CTL and μ-calculus specifications, and a 3EXPTIME upper bound for CTL* specifications.

[e]In the emptiness game, the states $V_{accept} = Q$ of the automaton become the states of player *accept*, while the set $V_{reject} = \{\delta(q, \sigma) \mid q \in Q, \sigma \in \Sigma\}$ of player reject consists of the different entries in the transition table of $\mathcal{C}$. Players *accept* ($|V| = |Q|$) and *reject* ($|V_{reject}| \leq |Q|^{|\Upsilon|}$) each have $n^{O(1)}$ states.

Theorem 4 *For a given architecture $A = (B, W, \mathcal{I}, \mathcal{O}, \{s_w \mid w \in W\})$ and a black-box process $b \in B$, checking resilient realizability and, if applicable, constructing a resilient realization $s_b : (2^{I_b})^* \to_r \mathfrak{D}_b$ can be performed in 2EXPTIME in the length of φ if φ is a CTL or μ-calculus specification, and in 3EXPTIME in the length of φ if φ is a CTL* specification.*

Proof. Following the construction described in Section 6.2, we construct a nondeterministic automaton $\mathcal{C}_\varphi$, which accepts exactly the resilient implementations of φ by b. If $n = |\varphi|$ denotes the length of the specification φ, then $\mathcal{C}_\varphi$ has

- $2^{n^{O(n)}}$ states and $n^{O(n)}$ colors if φ is a CTL specification,
- $2^{2^{2^{O(n)}}}$ states and $2^{2^{O(n)}}$ colors if φ is a CTL* specification, and
- $2^{n^{O(n^3)}}$ states and $n^{O(n^3)}$ colors if φ is a μ-calculus specification.

By Lemma 8 we can check the emptiness of $\mathcal{C}_\varphi$ and, if $\mathcal{C}_\varphi$ is non-empty, construct a regular tree accepted by $\mathcal{C}_\varphi$ in time polynomial in the number of states and exponential in the number of colors of $\mathcal{C}_\varphi$. □

6.5. Lower Bounds

To demonstrate that the upper bounds are sharp, we give a reduction from the synthesis problem in reactive environments with complete information, which is known to be 2EXPTIME and 3EXPTIME hard for CTL and CTL*, respectively [8]. In synthesis with reactive environments and complete information, we have only one process b, for which a (deterministic) strategy $s_b : (2^{O_{env}})^* \to_r \mathfrak{D}_b$ is sought. ($\mathfrak{D}_b$ is the set of singleton subsets of 2^{O_b}. The environment can react to the input by restricting its actions to a non-empty subset of its output variables O_{env}, which can be viewed as a non-deterministic strategy $s_{env} : (2^{O_{env} \cup O_b})^* \to 2^{2^{O_{env}}} \smallsetminus \{\emptyset\}$.) In our terms, a strategy $s_b : (2^{O_{env}})^* \to \mathfrak{D}_b$ implements a specification φ if, for all strategies $s_{env} : (2^{O_{env} \cup O_b})^* \to 2^{2^{O_{env}}} \smallsetminus \{\emptyset\}$ of the environment, $\langle \| s_b \otimes s_{env} \|, dir \rangle$ is a model of φ.

We encode this synthesis problem as the realizability of φ by b against a black-box process e with output O_e and an external environment without output. The second black-box process e plays the role of the reactive environment. Formally, we define the architecture $A = (\{b, e\}, \emptyset, env, \{I_b = O_e, I_e = I_{env} = V\}, \{O_{env} = \emptyset, O_b, O_e\}, \emptyset)$. The determinacy of s_b can be guaranteed by construction (by setting $\mathfrak{D}_b$ to the set of singleton subsets of 2^{O_b}). Alternatively, we can ensure the determinacy of s_b by strengthening the specification φ such that only deterministic strategies are allowed: We solve the realizability problem for $\varphi' = \varphi \wedge \psi$, where $\psi = \bigwedge_{o \in O_b} \mathrm{AG}\,((\mathrm{EX}o \to \mathrm{AX}o) \wedge (\mathrm{EX}\neg o \to \mathrm{AX}\neg o))$ (which is linear in φ).

Theorem 5 *The realizability problem $(A, \{b\})^{\exists} \models \varphi$ is 3EXPTIME complete for CTL* and 2EXPTIME complete for CTL and μ-calculus specifications in the length $|\varphi|$ of the specification.*

Proof. The lower bounds for CTL and CTL* follow from the equal lower bounds for the synthesis problem with reactive environments [8]. The lower bound for the μ-calculus is established by the lower bound for CTL. The upper bound is demonstrated by Theorem 4. $\square$

6.6. Universal Specifications

For universal specifications, the quantification step of Subsection 6.3.3 can be simplified, resulting in an exponential improvement in the complexity. A specification is universal if its models are recognized by a universal symmetric automaton. A symmetric automaton $\mathcal{U} = (\Sigma, Q, q_0, \delta, \alpha)$ is called *universal* if the mapping of the transition function δ contains no existential atoms (i.e., no atoms $(q, \Diamond) \in Q \times \{\Diamond\}$). The universal specifications include in particular the formulas of the following logics:

- the syntactic subsets ACTL* and ACTL of CTL* and CTL, respectively, which do not contain the existential path quantifier,
- the syntactic subset of the modal μ-calculus that does not contain the existential successor operator $\Diamond$, and
- trace languages like LTL.

Intuitively, the quantification step is used to guess, for a given strategy $s_b : (2^V)^* \to \mathfrak{D}_b$ of a process b, a strategy $s_{B \smallsetminus \{b\}} : (2^V)^* \to \mathfrak{D}_{B \smallsetminus \{b\}}$ for the remaining black-box processes $B \smallsetminus \{b\}$ for which φ_b does *not* hold, i.e., $\langle \| s_b \times s_W \times s_{B \smallsetminus \{b\}} \times s_{env} \|, dir \rangle \not\models \varphi_b$. If φ_b is a *universal* specification, the strategy can be set to the constant value $\bigoplus_{b \neq b' \in B} \{2^{O_{b'}}\}$.

Theorem 6 *If φ is a universal specification over a set AP of atomic propositions, $\langle Y, l : Y \to 2^{AP} \rangle \not\models \varphi$ is no model of φ, and $\langle Y', l' : Y' \to 2^{AP} \rangle$ is a 2^{AP}-labeled tree with $Y \subseteq Y'$ and $l(y) = l'(y)$ for all $y \in Y$, then $\langle Y', l' : Y' \to 2^{AP} \rangle$ is no model of φ.*

Proof. Let $\mathcal{U}_\varphi$ be a universal automaton recognizing φ and r a winning strategy for player *reject* in the acceptance game of $\langle Y, l : Y \to 2^{AP} \rangle$ (such a strategy exists, since $\langle Y, l : Y \to 2^{AP} \rangle \not\models \varphi$ is no model of φ). Then r is also a winning strategy for $\langle Y', l' : Y' \to 2^{AP} \rangle$ (r always picks the same atom and the same direction for $\langle Y', l' \rangle$ as for $\langle Y, l \rangle$, so $\langle Y, l \rangle$ is never left). $\square$

Consequently, we can assume w.l.o.g. that the strategies $s_{b'}$ of all remaining black-box processes $b' \in B \smallsetminus \{b\}$ are the constant strategies that constantly enable all directions.

It is worth noting that the simplification of the quantification step depends on the choice of the sets $\mathfrak{D}_{b'}$ of possible decisions. In Section 2, the set of possible decisions was fixed to $\mathfrak{D}_{b'} = 2^{2^{O_{b'}}} \smallsetminus \{\emptyset\}$, but a different choice like a restriction to deterministic decisions would not affect the argumentation for non-universal specifications. For universal specifications, the discussed simplification depends on the existence of a maximal element in $\mathfrak{D}_{b'}$.

Corollary 1 *If φ is a universal specification and, for all $b \neq b' \in B$, $\mathfrak{D}_{b'}$ has a maximal element $\overline{d}_{b'} \in \mathfrak{D}_{b'}$ ($\forall d \in \mathfrak{D}_{b'}. d \subseteq \overline{d}_{b'}$), than $\langle \| s_b \otimes s_W \otimes s_{B \smallsetminus \{b\}} \otimes$*

$s_{env}\|, dir\rangle \models \varphi$ *is a model of* φ *for all* $s_{B\smallsetminus\{b\}} : (2^V)^* \to \mathfrak{D}_{B\smallsetminus\{b\}}$ *iff* $\langle\|s_b \otimes s_W \otimes s_{B\smallsetminus\{b\}} \otimes s_{env}\|, dir\rangle \models \varphi$ *is a model of* φ *for* $s_{B\smallsetminus\{b\}} : (2^V)^* \to \{\bigoplus_{b\neq b'\in B} \overline{d}_{b'}\}$. □

For trace languages, it is safe to *extend* $\mathfrak{D}_{b'}$ by a maximal element $\overline{d}_{b'} = \bigcup_{d\in\mathfrak{D}_{b'}} d$, since no new traces are introduced by this extension (i.e., each trace in $\|s_b \otimes s_W \otimes s_{B\smallsetminus\{b\}} \otimes s_{env}\|$ for $\{s_{b'} : (2^V)^* \to \{\overline{s}_{b'}\} \mid b \neq b' \in B\}$ is a trace in $\|s_b \otimes s_W \otimes s_{B\smallsetminus\{b\}} \otimes s_{env}\|$ for some $\{s_{b'} : (2^V)^* \to \mathfrak{D}_{b'} \mid b \neq b' \in B\}$).

This is, however, *not* true for general universal specifications: if the set of black-box processes $B = \{b, b'\}$ consists of two processes b and b' and we allow only deterministic implementations of b', then b can resiliently realize $\mathrm{AG}\,(\mathrm{AX}p) \vee (\mathrm{AX}\neg p)$. Obviously, this is no longer true if we add a maximal element to $\mathfrak{D}_{b'}$, which simultaneously allows for successors where p holds and for successors where p does not hold.

6.7. Premise (S)

The correctness of Premise (S) can be checked along the same lines: we check whether the empty strategy resiliently realizes $\bigwedge_{b\in B} \varphi_b \to \psi$. Since $\mathfrak{D}_b = \{\emptyset\}$ and $I_b = \emptyset$, the automaton $\mathcal{B}_\varphi$ (with n states and c colors) is an alternating word automaton over the single-letter alphabet, whose emptiness can be checked in $n^{O(c)}$ time. Checking Premise (S) is therefore in EXPTIME for CTL and μ-calculus specifications and in 2EXPTIME for CTL* specifications, respectively, in $|\bigwedge\limits_{b\in B} \varphi_b \to \psi|$.

7. Conclusions

We have introduced a sound and complete proof rule for distributed synthesis, which reduces the distributed synthesis problem to the simpler task of deciding the *resilient realizability* of *local* process specifications.

Synthesizing resilient implementations generalizes the synthesis of open systems. Open synthesis assumes an environment with *maximal* behavior. For resilient implementations, this environment model is extended in two aspects: (1) The other black-box processes add a *reactive* component to the environment, and (2) the process only has *incomplete information* about the environment behavior.

Extension (1) turns out to be expensive. Adding the reactive component increases the complexity for CTL specifications from EXPTIME [9] to 2EXPTIME [8], and for CTL* specifications from 2EXPTIME [9] to 3EXPTIME [8]. As shown in Section 6, extension (2) has no extra cost. This settles an open question of [8]: The complexity of synthesizing a single process in a distributed architecture is still 2EXPTIME and 3EXPTIME complete, respectively.

The semi-automatic compositional approach is an efficient alternative to fully automatic distributed synthesis. Distributed synthesis is only decidable for a restricted class of architectures. For this class of architectures, the complexity is nonelementary [1, 2, 3].

The situation is similar to the *verification* of distributed systems, where the compositional approach is well-established [13]. Our proof rule is a first example of a compositional synthesis technique. The rule is complete and therefore sufficient to decompose any realizable specification. The rule may, however, be less convenient to use than some compositional verification rules that, for example, apply circular assume-guarantee reasoning [14]. Defining such rules for the synthesis problem is an interesting topic of future research.

Acknowledgments

We thank the reviewers for highly valuable comments on preliminary versions of this paper. This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

References

1. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. FOCS. (1990) 746–757
2. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: IEEE Symposium on Logic in Computer Science. (2001)
3. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. LICS, IEEE Computer Society Press (2005) 321–330
4. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. IBM Workshop on Logics of Programs. Volume 131 of LNCS., Springer-Verlag (1981) 52–71
5. Wolper, P.: Synthesis of Communicating Processes from Temporal-Logic Specifications. PhD thesis, Stanford University (1982)
6. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete informatio. In: Proc. 2nd International Conference on Temporal Logic (ICTL'97). (1997)
7. Kupferman, O., Vardi, M.Y.: μ-calculus synthesis. In: Proc. MFCS. Volume 1893 of LNCS., Springer-Verlag (2000) 497–507
8. 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
9. Kupferman, O., Vardi, M.Y.: Church's problem revisited. The bulletin of Symbolic Logic **5** (1999) 245–263
10. Muller, D.E., Schupp, P.E.: Alternating automata on infinite trees. Theor. Comput. Sci. **54** (1987) 267–276
11. 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** (1995) 69–107
12. Jurdziński, M.: Small progress measures for solving parity games. In: Proc. STACS. Volume 1770 of LNCS., Springer-Verlag (2000) 290–301
13. de Roever, W.P., Langmaack, H., Pnueli, A., eds.: Compositionality: The Significant Difference. COMPOS'97. Volume 1536 of LNCS., Springer Verlag (1998)
14. Maier, P.: A Lattice-Theoretic Framework For Circular Assume-Guarantee Reasoning. PhD thesis, Universität des Saarlandes, Saarbrücken (2003)
15. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proc. FOCS, IEEE Computer Society Press (1991) 368–377

Appendix A: Logics

This appendix provides a brief description of the logics used in the paper:

- CTL* and its sublogic ACTL*, LTL, CTL, and ACTL, and
- the modal μ-calculus.

These logics are interpreted over total 2^{AP}-trees where AP denotes the set of atomic propositions. The considered logics have the finite model property: if they accept some tree, they accept a regular tree, i.e., a tree which can be constructed by unraveling a finite transducer (cf. [4, 5, 15]). The finite model property holds both for general total trees and for full Υ-trees with a predefined set of directions Υ.

1. Branching-Time Logics

Syntax. CTL* distinguishes state and path formulas. Let AP denote the set of atomic propositions.

- *true* and *false* are state formulas.
- p and $\neg p$ and state formulas for all $p \in AP$.
- If φ and ψ are state formulas then $\varphi \wedge \psi$ and $\varphi \vee \psi$ are state formulas.
- If π is a path formula then $\mathrm{A}\pi$ and $\mathrm{E}\pi$ are state formulas.
- Every state formula is also a path formula.
- If π and τ are path formulas then
  - $\pi \wedge \tau$ and $\pi \vee \tau$,
  - $\mathrm{X}\pi$, $\mathrm{G}\pi$ and $\mathrm{F}\pi$, and
  - $\pi\mathrm{U}\tau$ and $\pi\mathrm{R}\tau$

  are path formulas.

Every state formula is a CTL* formula.

Semantics. A CTL* specification with atomic propositions AP is interpreted over 2^{AP}-labeled trees. The paths of a total Υ-tree $Y \subseteq \Upsilon^*$ rooted in a node $y \in Y$ are defined as

$$\mathit{paths}(y) = \{y_0, y_1, y_2, y_3, \cdots \mid y_0 = y \wedge \forall n \in \mathbb{N}_0 \exists \upsilon \in \Upsilon .\, y_{n+1} = y_n \cdot \upsilon\}.$$

The paths of Y is the set of all paths rooted in some $y \in Y$:

$$\mathit{paths}(Y) = \bigcup \{\mathit{paths}(y) \mid y \in Y\}.$$

A CTL* specification is evaluated along the structure of a specification. For a state formula φ and a total tree 2^{AP}-labeled tree $\langle Y, l\rangle$, $\|\varphi\|_{\langle Y,l\rangle} \subseteq Y$ denotes the set of nodes, where φ holds. A total tree 2^{AP}-labeled tree $\langle Y, l\rangle$ is a model of φ iff φ holds on the root $\varepsilon \in Y$ of Y.

- Atomic propositions are interpreted as follows:
  $\|\mathit{false}\|_{\langle Y,l\rangle} = \emptyset$ and $\|\mathit{true}\|_{\langle Y,l\rangle} = Y$; and
  $\|p\|_{\langle Y,l\rangle} = \{y \in Y \mid p \in l(y)\}$ and $\|\neg p\|_{\langle Y,l\rangle} = \{y \in Y \mid p \notin l(y)\}$.

- Conjunction and disjunction are interpreted as intersection and union, respectively:

  $\|\varphi \wedge \psi\|_{\langle Y,l\rangle} = \|\varphi\|_{\langle Y,l\rangle} \cap \|\psi\|_{\langle Y,l\rangle}$ and $\|\varphi \vee \psi\|_{\langle Y,l\rangle} = \|\varphi\|_{\langle Y,l\rangle} \cup \|\psi\|_{\langle Y,l\rangle}$.

- Formulas starting with a universal or existential path quantifier hold true in a node y, if the respective path formula holds true for some or all paths rooted in y, respectively:

  $\|\mathrm{A}\varphi\|_{\langle Y,l\rangle} = \{y \in Y \mid \forall \pi \in \mathit{paths}(y).\, \pi \in \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle}$, and

  $\|\mathrm{E}\varphi\|_{\langle Y,l\rangle} = \{y \in Y \mid \exists \pi \in \mathit{paths}(y).\, \pi \in \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle}$.

For a path formula φ and a total tree 2^{AP}-labeled tree $\langle Y, l\rangle$, $\|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} \subseteq \mathit{paths}(Y)$ denotes the set of paths of Y where φ holds. Path formulas are interpreted as follows:

- For state formulas φ, $\|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \bigcup\{\mathit{paths}(y) \mid y \in \|\varphi\|_{\langle Y,l\rangle}\}$.

- Conjunction and disjunction are interpreted as intersection and union, respectively:

  $\|\varphi \wedge \psi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} \cap \|\psi\|^{\mathit{paths}}_{\langle Y,l\rangle}$ and $\|\varphi \vee \psi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} \cup \|\psi\|^{\mathit{paths}}_{\langle Y,l\rangle}$.

- A path $\pi = y_0, y_1, y_2, y_3 \cdots$ satisfies $\mathrm{X}\varphi$ (read: next φ), if the path $y_1, y_2, y_3 \cdots$ obtained by deleting the first letter of π satisfies φ:

  $\|\mathrm{X}\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \{y_0, y_1, y_2, y_3 \cdots \in \mathit{paths}(Y) \mid y_1, y_2, y_3 \cdots \in \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle}$.

- A path $\pi = y_0, y_1, y_2, y_3 \cdots$ satisfies $\mathrm{G}\varphi$ (read: globally φ), if every path $y_n, y_{n+1}, y_{n+2}, y_{n+3} \cdots$ obtained by deleting some (possibly empty) initial sequence $y_0, y_1, y_2, y_3 \cdots y_{n-1}$ of π satisfies φ:

  $\|\mathrm{G}\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \{y_0, y_1, y_2 \cdots \in \mathit{paths}(Y) \mid \forall n \in \mathbb{N}_0.\, y_n, y_{n+1}, y_{n+2} \cdots \in \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle}$.

- A path $\pi = y_0, y_1, y_2, y_3 \cdots$ satisfies $\mathrm{F}\varphi$ (read: finally φ), if some path $y_n, y_{n+1}, y_{n+2}, y_{n+3} \cdots$ obtained by deleting some (possibly empty) initial sequence $y_0, y_1, y_2, y_3 \cdots y_{n-1}$ of π satisfies φ:

  $\|\mathrm{F}\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle} = \{y_0, y_1, y_2 \cdots \in \mathit{paths}(Y) \mid \exists n \in \mathbb{N}_0.\, y_n, y_{n+1}, y_{n+2} \cdots \in \|\varphi\|^{\mathit{paths}}_{\langle Y,l\rangle}$.

- A path $\pi = y_0, y_1, y_2, y_3 \cdots$ satisfies $\varphi \mathrm{U} \psi$ (read: φ until ψ), if there is a natural number $n \in \mathbb{N}_0$ such that

  (1) the path $y_n, y_{n+1}, y_{n+2}, y_{n+3} \cdots$ obtained by deleting the initial sequence $y_0, y_1, y_2, y_3 \cdots y_{n-1}$ of π satisfies φ, and

  (2) for all $i \leq n$, the path $y_i, y_{i+1}, y_{i+2}, y_{i+3} \cdots$ obtained by deleting the initial sequence $y_0, y_1, y_2, y_3 \cdots y_{i-1}$ of π satisfies ψ:

  $\|\varphi \mathrm{U} \psi\|^{paths}_{\langle Y,l \rangle} = \{y_0, y_1, y_2, y_3 \cdots \in paths(Y) \mid$

  $\exists n \in \mathbb{N}_0. \, (y_n, y_{n+1}, y_{n+2} \cdots \in \|\varphi\|^{paths}_{\langle Y,l \rangle} \wedge \forall i \leq n. \, y_i, y_{i+1}, y_{i+2} \cdots \in \|\psi\|^{paths}_{\langle Y,l \rangle})\}.$

- A path $\pi = y_0, y_1, y_2, y_3 \cdots$ satisfies $\varphi \mathrm{R} \psi$ (read: φ releases ψ), if for every natural number $n \in \mathbb{N}_0$

  (1) the path $y_n, y_{n+1}, y_{n+2}, y_{n+3} \cdots$ obtained by deleting the initial sequence $y_0, y_1, y_2, y_3 \cdots y_{i-1}$ of π satisfies φ, or

  (2) there is an $i \leq n$, such that the path $y_i, y_{i+1}, y_{i+2}, y_{i+3} \cdots$ obtained by deleting the initial sequence $y_0, y_1, y_2, y_3 \cdots y_{n-1}$ of π satisfies ψ:

  $\|\varphi \mathrm{R} \psi\|^{paths}_{\langle Y,l \rangle} = \{y_0, y_1, y_2, y_3 \cdots \in paths(Y) \mid$

  $\forall n \in \mathbb{N}_0. \, (y_n, y_{n+1}, y_{n+2} \cdots \in \|\psi\|^{paths}_{\langle Y,l \rangle} \vee \exists i \leq n. \, y_i, y_{i+1}, y_{i+2} \cdots \in \|\varphi\|^{paths}_{\langle Y,l \rangle})\}.$

Sublogics CTL is the sublogic of CTL* where each path operator (X, G, F, U and R) is directly preceded by a path quantifier (A or E). When considering universal specifications, we also refer to the sublogics ACTL / ACTL* is the syntactical sublogic of CTL / CTL*, where no existential path quantifiers (E) appear. LTL is the syntactical sublogic of ACTL* where (universal) path quantifier appear only as the first sign of a specification. Usually this quantification is left implicit.

2. μ-Calculus

The *μ-calculus* is a modal fixed-point logic which is (like CTL*) interpreted over total 2^{AP}-labeled trees. It is more expressive than CTL and CTL*.

μ-Calculus Syntax. Let AP and B denote disjoint finite sets of atomic propositions and bound variables, respectively. Then,

- *true* and *false* are μ-calculus formulas.
- p, $\neg p$ and x are μ-calculus formulas for all $p \in AP$ and $x \in B$.
- If φ and ψ are μ-calculus formulas then $\varphi \wedge \psi$ and $\varphi \vee \psi$ are μ-calculus formulas.
- If φ is an μ-calculus formula then $\Box \varphi$ and $\Diamond \varphi$ are μ-calculus formulas.
- If $x \in B$ and φ is an μ-calculus formula where x occurs only free, then $\mu x . \varphi$ and $\nu x . \varphi$ are μ-calculus formulas.

μ-Calculus Semantics. A μ-calculus formula φ with atomic propositions AP is interpreted over a total 2^{AP}-labeled tree $\langle Y, l\rangle$. $\|\varphi\|_{\langle Y,l\rangle} \subseteq Y$ denotes the nodes of Y where φ holds. A total 2^{AP}-labeled tree $\langle Y, l\rangle$ is a *model* of a specification φ with atomic propositions AP iff φ holds in its root ($\varepsilon \in \|\varphi\|_{\langle Y,l\rangle}$).

- Atomic propositions are interpreted as follows:
  $\|\mathit{false}\|_{\langle Y,l\rangle} = \emptyset$ and $\|\mathit{true}\|_{\langle Y,l\rangle} = Y$,
  $\|p\|_{\langle Y,l\rangle} = \{y \in Y \mid p \in l(y)\}$ and $\|\neg p\|_{\langle Y,l\rangle} = \{y \in Y \mid p \notin l(y)\}$.

- Conjunction and disjunction are interpreted as intersection and union, respectively:

  $\|\varphi \wedge \psi\|_{\langle Y,l\rangle} = \|\varphi\|_{\langle Y,l\rangle} \cap \|\psi\|_{\langle Y,l\rangle}$ and $\|\varphi \vee \psi\|_{\langle Y,l\rangle} = \|\varphi\|_{\langle Y,l\rangle} \cup \|\psi\|_{\langle Y,l\rangle}$.

- A node $y \in Y$ is in $\|\Box\varphi\|_{\langle Y,l\rangle}$ iff φ holds in all successor states:
  $\|\Box\varphi\|_{\langle Y,l\rangle} = \{y \in Y \mid \forall \upsilon \in \Upsilon.\, y \cdot \upsilon \in Y \rightarrow y \cdot \upsilon \in \|\varphi\|_{\langle Y,l\rangle}\}$.

- A node $y \in Y$ is in $\|\Diamond\varphi\|_{\langle Y,l\rangle}$ iff φ holds in some successor state:
  $\|\Diamond\varphi\|_{\langle Y,l\rangle} = \{y \in Y \mid \exists \upsilon \in \Upsilon.\, y \cdot \upsilon \in \|\varphi\|_{\langle Y,l\rangle}\}$.

- The least and greatest fixed points are interpreted as follows:

  $\|\mu x.\varphi\|_{\langle Y,l\rangle} = \bigcap\{Y_x \subseteq Y \mid \|\varphi\|_{\langle Y,l_x^{Y_x}\rangle} \subseteq Y_x\}$, and
  $\|\nu x.\varphi\|_{\langle Y,l\rangle} = \bigcup\{Y_x \subseteq Y \mid \|\varphi\|_{\langle Y,l_x^{Y_x}\rangle} \supseteq Y_x\}$,

  where $\langle Y, l_x^{Y_x}\rangle$ denotes the tree with the modified labeling function $l_x^{Y_x} : Y \rightarrow 2^{AP \cup \{x\}}$ with

  - $l_x^{Y_x}(y) \cap AP = l(y)$ and
  - $x \in l_x^{Y_x}(y) \Leftrightarrow y \in Y_x \subseteq Y$.

  Since the bound variable x occurs only positively in φ, $\|\varphi\|_{\langle Y,l_x^{Y_x}\rangle}$ is monotone in Y_x and the fixed points are well-defined.