Cooperative Reactive Synthesis[*]

Roderick Bloem

IAIK, Graz University of
Technology, Graz, Austria

Rüdiger Ehlers

University of Bremen and
DFKI GmbH, Bremen, Germany

Robert Könighofer

IAIK, Graz University of
Technology, Graz, Austria

A modern approach to engineering correct-by-construction systems is to synthesize them automatically from formal specifications. Oftentimes, a system can only satisfy its guarantees if certain environment assumptions hold, which motivates their inclusion in the system specification. Experience with modern synthesis approaches shows that synthesized systems tend to satisfy their specifications by actively working towards the violation of the assumptions rather than satisfying assumptions and guarantees together. Such uncooperative behavior is undesirable because it violates the aim of synthesis: the system should try to satisfy its guarantees and use the assumptions only when needed. Also, the assumptions often describe the valid behavior of other components in a bigger system, which should not be obstructed unnecessarily.

In this paper, we present a hierarchy of *cooperation levels* between system and environment. Each level describes how well the system enforces both the assumptions and guarantees. We show how to synthesize systems that achieve the highest possible cooperation level for a given specification in Linear Temporal Logic (LTL). The synthesized systems can also exploit cooperative environment behavior during operation to reach a higher cooperation level that is not enforceable by the system initially. The worst-case time complexity of our synthesis procedure is doubly-exponential, which matches the complexity of standard LTL synthesis. This is an extended version of [7] that features an additional appendix.

1 Introduction

When synthesizing reactive systems from their formal specifications, we typically start with a set of guarantees that the system should fulfill and a set of assumptions about the environment. A synthesis tool then computes an implementation that satisfies the guarantees in all environments that satisfy the assumptions. In many specifications, the system can influence whether the assumptions are satisfied; in particular, the system can actively force the environment to violate the assumptions. The resulting implementation is correct: it fulfills its guarantees if the assumptions are fulfilled. However, it does so in an undesirable way.

Take for example a flight stabilization system that adds small forces to choices of forces issued by the pilot. The system guarantees state that there is little jitter in the absolute forces applied. This can only work for "well-behaved" evolutions of the manually selected force values, which in turn depend on the forces currently applied. Without the well-behavedness assumption, the system has no way to stabilize the flight. However, if the system has the opportunity to add forces that make the pilot violate this assumption, it can do so without violating its overall specification (namely that the guarantees must hold whenever the assumptions hold). Clearly, such behavior is not covered by the specifier's intent.

This observation leads to the question of how we can synthesize systems that cooperate with the environment whenever possible. Our main idea is that, at any point in time, the system should try to be

---

as cooperative as possible while still ensuring correctness. Depending on the concrete situation, several levels of cooperation may be possible. Some examples are:

1. The system can enforce both the assumptions and guarantees to hold.

2. The system can enforce the guarantees if the assumptions hold and the system can give the environment the opportunity to satisfy the assumptions at the same time.

3. The system can neither enforce the assumptions nor the guarantees, but environment and system together can satisfy both.

The first of these levels is most beneficial: there is no need to rely on the environment, not even for satisfying the assumptions. On the second level we can satisfy the correctness objective even without allowing the system to enforce an assumption violation. From states of the third level, the system cannot enforce correctness. However, instead of resigning and behaving arbitrarily, the system still offers some executions along which both the assumptions and guarantees are fulfilled, thereby optimistically assuming that the environment is helpful.

In this paper, we perform a rigorous analysis of cooperation between the system and its environment in the setting of reactive synthesis. We generalize the three levels of cooperation from above to a fine-grained *cooperation hierarchy* that allows us to reason about how cooperative a controller for a given specification can be. A level in our hierarchy describes what objectives (in terms of assumptions and guarantees) the controller can achieve on its own and for what objectives it has to rely on the environment.

As a second contribution, we present a synthesis procedure to construct a controller that always picks the highest possible cooperation level in our hierarchy for a given linear-time temporal logic (LTL) specification. The synthesized controllers do not only enforce the highest possible level statically, but also exploit environment behavior that enables reaching a higher level that cannot be enforced initially. Thus, implementations synthesized with our approach are *maximally cooperative*, without any need to declare cooperation in the specification explicitly by enumerating scenarios in which a system can operate in a cooperative way. Our *maximally cooperative synthesis procedure* takes at most doubly-exponential time, which matches the complexity of standard LTL synthesis. Our techniques are applicable to all ω-regular word languages. For specifications given as deterministic Rabin word automata, the complexity is polynomial in the number of states and exponential in the number of acceptance condition pairs.

Outline. The presentation of this paper is structured as follows. Section 2 reviews related work and Section 3 introduces background and notation. Section 4 then presents our hierarchy of cooperation levels, while Section 5 describes our synthesis approach for obtaining cooperative systems. We conclude in Section 6.

2   Related Work

In earlier work [6], we already pointed out that existing synthesis approaches handle environment assumptions in an unsatisfactory way. We postulated that synthesized systems should (1) *be correct*, (2) *not be lazy* by satisfying guarantees even if assumptions are violated, (3) *never give up* by working towards the satisfaction of the guarantees even if this is not possible in the worst case, and (4) *cooperate* by helping the environment to satisfy the assumptions that we made about it [6]. Our current work addresses all of these challenges.

Besides correctness, our main focus is on cooperation, which is also addressed by Assume-Guarantee Synthesis [12] and synthesis under rationality assumptions [3, 10, 11, 18]. These approaches do not distinguish between system and environment, but synthesize implementations for both. Each component

works under certain assumptions about the other components not deviating from their synthesized implementations arbitrarily. In contrast, our work handles the system and its environment asymmetrically: we prefer the guarantees over the assumptions and prioritize correctness over supporting the environment.

Similar to existing work on synthesis of robust [4] and error-resilient [16] systems, our synthesis approach is also *not lazy* [6] in satisfying guarantees. The reason is that satisfying guarantees is more preferable in our cooperation hierarchy than satisfying guarantees only if the assumptions are satisfied. In contrast to the existing work, we do not only consider the worst case environment behavior for satisfying guarantees (even if assumptions are violated), but also the scenario where guarantees can only be satisfied with the help of the environment.

Like [17], we also address the *never give up* challenge [6] because our cooperation hierarchy also includes levels on which the system cannot enforce the guarantees any more. Still, our synthesis approach lets the system satisfy the guarantees for *some* environment behavior whenever that is possible.

In contrast to quantitative synthesis [1,5] where the synthesized system maximizes a certain pay-off, our approach is purely qualitative: a certain level of the hierarchy is either achieved or not. In this way, we not only avoid the computational blow-up induced by operating with numerical data, but also remove the need to assign meaningful quantities (such as probabilities) to the specifications.

Finally, [15] presents an extension of a synthesis algorithm for so-called GR(1) specifications [8] to produce mission plans for robots that always offer some execution on which both assumptions and guarantees are satisfied. Hence, [15] gives a solution for one level in our hierarchy, implemented for a subset of LTL.

3 Preliminaries

We denote the Boolean domain by $\mathbb{B} = \{\text{false}, \text{true}\}$ and the set of natural numbers (including 0) by $\mathbb{N}$.

Words: We consider synthesis of reactive systems with a finite set $I = \{i_1, \ldots, i_m\}$ of Boolean input signals and a finite set $O = \{o_1, \ldots, o_n\}$ of Boolean outputs. The input alphabet is $\mathscr{I} = 2^I$, the output alphabet is $\mathscr{O} = 2^O$, and $\Sigma = \mathscr{I} \times \mathscr{O}$. The set of finite (infinite) words over Σ is denoted by Σ^* (Σ^ω), where ε is the word of length 0. A set $L \subseteq \Sigma^\omega$ of infinite words is called a (*word*) *language*.

Specifications: A specification φ over Σ defines a language $L(\varphi)$ of allowed words. In this paper, specifications consist of two parts, the *environment assumptions* $\mathscr{A}$ and the *system guarantees* $\mathscr{G}$. These parts can be combined by logical operators. For example, we write $\mathscr{A} \to \mathscr{G}$ to specify that the guarantees (only) need to hold if the assumptions are satisfied. We say that some finite word $w \in \Sigma^*$ is a *bad prefix* for φ if there does not exist a word $w' \in \Sigma^\omega$ such that $ww' \in L(\varphi)$. The set of infinite words that have no bad prefixes of φ will be called the *safety hull* of φ.

Reactive systems: A reactive system interacts with its environment in a synchronous way. In every time step j, the system first provides an output letter $y_j \in \mathscr{O}$, after which the environment responds with an input letter $x_j \in \mathscr{I}$. This is repeated in an infinite execution to produce the *trace* $w = (x_0, y_0)(x_1, y_1) \ldots \in \Sigma^\omega$.

We can represent the complete behavior of a reactive system by a *computation tree* $\langle T, \tau \rangle$, where T is the set of nodes and a subset of $\mathscr{I}^*$, and τ assigns labels to the tree nodes. In such a tree, we have $\tau : T \to \mathscr{O}$, i.e., the tree nodes are labeled by the last output of the system. We call trees $\langle T, \tau \rangle$ with $T = \mathscr{I}^*$ *full trees* and consider only these henceforth, unless otherwise stated. We say that some trace $w = (x_0, y_0)(x_1, y_1) \ldots \in \Sigma^\omega$ is *included* in $\langle T, \tau \rangle$ if for all $i \in \mathbb{N}$, $y_i = \tau(x_0 \ldots x_{i-1})$. In such a case, we also say that w is a *branch* of $\langle T, \tau \rangle$.

We say that some specification φ is *realizable* if there exists a computation tree for the system all of whose traces are in $L(\varphi)$. The set of traces of $\langle T, \tau \rangle$ is denoted by $L(\langle T, \tau \rangle)$ and called the *word language* of $\langle T, \tau \rangle$. The set of all computation trees over $\mathscr{I}$ and $\mathscr{O}$ is denoted by $\mathscr{S}$. Since a computation tree represents the full behavior of a reactive system, we use these two terms interchangeably.

Rabin word and tree automata: In order to represent specifications over Σ in a finitary way, we employ *Rabin automata*. For word languages, we use deterministic Rabin word automata, which are defined as tuples $\mathscr{R} = (Q, \Sigma, \delta, q_0, \mathscr{F})$, where Q is the finite set of *states*, Σ is the finite alphabet, $\delta : Q \times \Sigma \to Q$ is the *transition function*, $q_0 \in Q$ is the initial state of $\mathscr{R}$, and $\mathscr{F} \subseteq 2^Q \times 2^Q$ is the *acceptance* condition of $\mathscr{R}$. Given a word $w \in \Sigma^\omega$, $\mathscr{R}$ induces a *run* $\pi = \pi_0 \pi_1 \pi_2 \ldots \in Q^\omega$, where $\pi_0 = q_0$ and for every $i \in \mathbb{N}$, we have $\pi_{i+1} = \delta(\pi_i, w_i)$. The run π is *accepting* if there exists some *acceptance condition pair* $(F, G) \in \mathscr{F}$ such that $\inf(\pi) \cap F = \emptyset$, and $\inf(\pi) \cap G \neq \emptyset$, where inf maps a sequence to the set of elements that appear infinitely often in the sequence. We say that w is accepted by $\mathscr{R}$ if there exists a run for w that is accepting. The language of $\mathscr{R}$, denoted as $L(\mathscr{R})$, is defined to be the set of words accepted by $\mathscr{R}$. The language of a state $q \in Q$ is defined to be the language of the automaton $\mathscr{R}_q = (Q, \Sigma, \delta, q, \mathscr{F})$, which differs from $\mathscr{R}$ only by its initial state. Without loss of generality, we assume that every Rabin automaton has a designated state $\top$ that has the full language, i.e., for which $L(\mathscr{R}_\top) = \Sigma^\omega$.

In addition to deterministic Rabin word automata, we will later also need non-deterministic Rabin tree automata. A non-deterministic Rabin tree automaton $\mathscr{R} = (Q, \mathscr{I}, \mathscr{O}, \delta, q_0, \mathscr{F})$ is defined similarly to a word automaton, except that δ has a different structure, and we now have two alphabets, namely a *branching alphabet* $\mathscr{I}$ and a *label alphabet* $\mathscr{O}$. The transition function δ is defined as $\delta : Q \times \mathscr{O} \to 2^{\mathscr{I} \to Q}$. A tree automaton accepts or rejects computation trees instead of words. Given a computation tree $\langle T, \tau \rangle$, we say that some tree $\langle T', \tau' \rangle$ is a *run tree* of $\mathscr{R}$ for $\langle T, \tau \rangle$ if $T' = T$, $\tau' : T' \to Q$, $\tau'(\varepsilon) = q_0$, and for all $t' \in T'$, there exists a function $f \in \delta(\tau'(t'), \tau(t'))$ such that for all $i \in \mathscr{I}$, $f(i) = \tau'(t'i)$. We say that $\langle T', \tau' \rangle$ is *accepting* if every branch of $\langle T', \tau' \rangle$ is accepting, i.e., its sequence of labellings satisfies the Rabin acceptance condition $\mathscr{F}$. We say that a tree automaton accepts a computation tree if it has a corresponding accepting run tree.

Given a Rabin word automaton $\mathscr{R}$ as specification over the alphabet $\Sigma = \mathscr{I} \times \mathscr{O}$, we can check if there exists a tree with branching alphabet $\mathscr{I}$ and label alphabet $\mathscr{O}$ all of whose traces are in the language of $\mathscr{R}$. It has been shown that this operation can be performed in time exponential in $|\mathscr{F}|$ and polynomial in $|Q|$ by first translating the Rabin word automaton to a non-deterministic Rabin tree automaton with the same set of states and the same acceptance condition (in linear time), and then checking the tree automaton's language for emptiness [19]. This approach gives rise to a reactive synthesis procedure for specifications in *linear temporal logic* (LTL) [19]: we can first translate the LTL specification to a deterministic Rabin word automaton with a number of states that is doubly-exponential in the length of the specification, and a number of acceptance condition pairs that is exponential in the length of the specification. Overall, this gives a doubly-exponential time procedure for LTL reactive synthesis, which matches the known complexity of the problem.

4   A Hierarchy of Cooperation Levels

In this section, we develop a hierarchy of cooperation levels that a system may achieve. We use a special logic to specify desired properties of the system to be synthesized. The syntax of a formula Φ in this logic is defined as

$$\Phi ::= \varphi \mid \langle E \rangle \, \varphi \mid \mathsf{G} \langle E \rangle \, \varphi \mid \Phi \wedge \Phi, \tag{1}$$

where φ is a linear-time specification (e.g., an LTL formula) over an alphabet Σ. The Boolean connective $\wedge$ has its expected semantics. The formula $\langle E \rangle \varphi$ is satisfied for a system $\langle T, \tau \rangle$ if there exists some trace of $\langle T, \tau \rangle$ on which φ holds. Similarly, the formula $\mathsf{G}\langle E \rangle \varphi$ is satisfied if, from any point in a reactive system's trace that has been seen so far, there exists some suffix trace of the system such that φ holds. We call an instance of the grammar in Formula 1 a *cooperation level specification*. Our logic ranges over computation trees and has similarities to *strategy logic* [13] as well as *alternating-time temporal logic* (ATL) [2]. Yet, its semantics, to be given below, is very different. In particular, linear-time specifications φ are seen as atomic and even within the scope of a G operator, φ is only evaluated on *complete traces* of a system, always starting at the root of a computation tree.

More formally, we define the semantics as follows. Formulas are interpreted over a computation tree $\langle T, \tau \rangle \in \mathscr{S}$ using the following rules:

$$
\begin{aligned}
\langle T, \tau \rangle &\models \varphi && \text{iff} \quad \forall w \in L(\langle T, \tau \rangle) : w \in L(\varphi) \\
\langle T, \tau \rangle &\models \langle E \rangle \varphi && \text{iff} \quad \exists w \in L(\langle T, \tau \rangle) : w \in L(\varphi) \\
\langle T, \tau \rangle &\models \mathsf{G}\langle E \rangle \varphi && \text{iff} \quad \forall w \in L(\langle T, \tau \rangle), i \in \mathbb{N} . \exists w' \in L(\langle T, \tau \rangle) \cap L(\varphi) . \\
& && \qquad w_0 \ldots w_{i-1} = w'_0 \ldots w'_{i-1} \\
\langle T, \tau \rangle &\models \Phi_1 \wedge \Phi_2 && \text{iff} \quad (\langle T, \tau \rangle \models \Phi_1) \wedge (\langle T, \tau \rangle \models \Phi_2)
\end{aligned}
$$

Most cases are straightforward, but $\mathsf{G}\langle E \rangle \varphi$ requires some attention. We follow an arbitrary trace of $\langle T, \tau \rangle$ up to step $i-1$. At step $i-1$, there must be some continuation of the trace that satisfies φ and that is part of the system's computation tree. If such a continuation exists in every step i, then $\mathsf{G}\langle E \rangle \varphi$ is satisfied.

4.1 Defining the Interesting Levels of Cooperation

We start with the linear-time specification $\mathscr{A}$, which represents the assumptions about the environment, and $\mathscr{G}$, which represents the guarantees. We can combine them to the linear-time properties $\mathscr{A} \to \mathscr{G}$ and $\mathscr{A} \wedge \mathscr{G}$. The first of these represents the classical correctness requirement for reactive synthesis, while the latter represents the optimistic linear-time property that both assumptions and guarantees hold along a trace of the system. As generators for cooperation level specifications, we consider all rules from the grammar in Equation 1 except for the second one, as it is rather weak, and only considers what can happen from the initial state of a system onwards. Additionally, leaving out the conjuncts of the form $\langle E \rangle \varphi$ strengthens the semantic foundation for our *maximally cooperative synthesis* approach in Sect. 5.2. However, we discuss the consideration of conjuncts of the form $\langle E \rangle \varphi$ in Section 4.3.

Combining all four linear-time properties with the two chosen ways of lifting a linear-time property to our logic gives the following different conjuncts for cooperation level specifications:

$$D = \{\mathscr{A} \to \mathscr{G}, \mathscr{G}, \mathscr{A}, \mathsf{G}\langle E \rangle (\mathscr{A} \wedge \mathscr{G}), \mathsf{G}\langle E \rangle \mathscr{G}, \mathsf{G}\langle E \rangle \mathscr{A}, \mathsf{G}\langle E \rangle (\mathscr{A} \to \mathscr{G})\} \tag{2}$$

A *cooperation level specification* is a conjunction between elements from this set. So there are $2^7 = 128$ possible cooperation levels in this setting. Note that we removed the linear-time property $\mathscr{A} \wedge \mathscr{G}$ from D, as it can be simulated by a conjunction between $\mathscr{A}$ and $\mathscr{G}$ on the level of cooperation level specifications.

We can reduce the 128 possible cooperation levels substantially by using our knowledge of the semantics of cooperation level specifications, which can be expressed in *reduction rules*. For example, if in a cooperation level specification, for some linear-time property $\varphi \in \{\mathscr{A}, \mathscr{G}, \mathscr{A} \to \mathscr{G}, \mathscr{A} \wedge \mathscr{G}\}$, φ is a conjunct along with $\mathsf{G}\langle E \rangle \varphi$, then we can remove the latter from the cooperation level specification as

$\mathscr{G} \wedge \mathscr{A}$

$\mathscr{A} \wedge \mathsf{G}\langle E\rangle\mathscr{G}$ 　　$\mathscr{G} \wedge \mathsf{G}\langle E\rangle\mathscr{A}$

$\mathscr{A}$ 　　$(\mathscr{A} \rightarrow \mathscr{G}) \wedge \mathsf{G}\langle E\rangle\mathscr{A}$ 　$\mathscr{G}$

$\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$ 　　$(\mathscr{A} \rightarrow \mathscr{G}) \wedge \mathsf{G}\langle E\rangle\mathscr{G}$

$(\mathsf{G}\langle E\rangle\mathscr{A}) \wedge (\mathsf{G}\langle E\rangle\mathscr{G})$

$(\mathsf{G}\langle E\rangle\mathscr{A}) \wedge \mathsf{G}\langle E\rangle(\mathscr{A} \rightarrow \mathscr{G})$

$\mathsf{G}\langle E\rangle\mathscr{A}$ 　　$\mathsf{G}\langle E\rangle\mathscr{G}$ 　$\mathscr{A} \rightarrow \mathscr{G}$

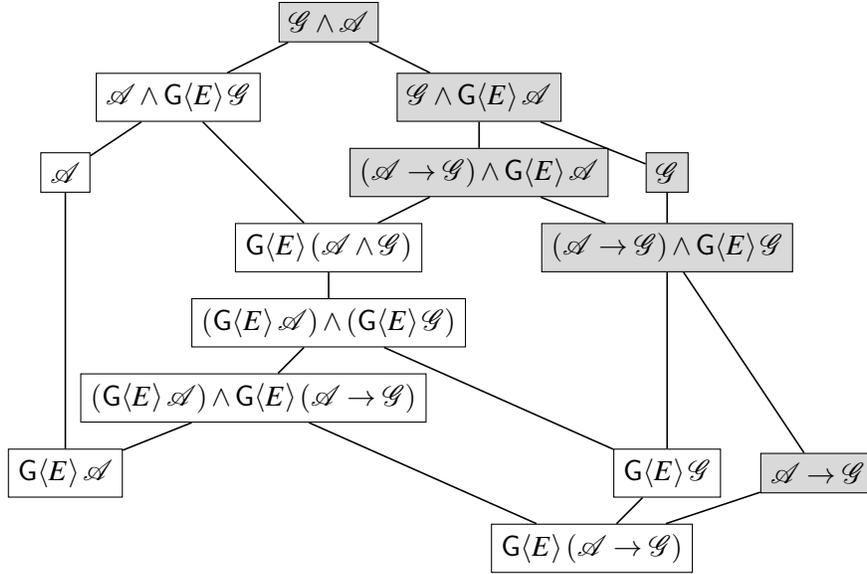$\mathsf{G}\langle E\rangle(\mathscr{A} \rightarrow \mathscr{G})$

Figure 1: A Hasse diagram of the hierarchy of cooperation levels.

that conjunct is implied by the former. This is because if along every trace of a computation tree, φ holds, then for every node in the tree, we can find a trace containing the node along with φ holds. In a similar fashion, we can observe

- that $\mathscr{G}$ implies $\mathscr{A} \rightarrow \mathscr{G}$,
- that $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$ implies $\mathsf{G}\langle E\rangle\mathscr{A}$ and $\mathsf{G}\langle E\rangle\mathscr{G}$,
- that $\mathsf{G}\langle E\rangle\mathscr{G}$ implies $\mathsf{G}\langle E\rangle(\mathscr{A} \rightarrow \mathscr{G})$,
- that $\mathscr{A} \rightarrow \mathscr{G}$ and $\mathscr{A}$ together imply $\mathscr{G}$, and
- that $\mathscr{A} \rightarrow \mathscr{G}$ and $\mathsf{G}\langle E\rangle\mathscr{A}$ together imply $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$,
- that $\mathscr{A}$ and $\mathsf{G}\langle E\rangle\mathscr{G}$ together imply $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$,
- that $\mathsf{G}\langle E\rangle(\mathscr{A} \rightarrow \mathscr{G})$ and $\mathscr{A}$ together imply $\mathsf{G}\langle E\rangle(\mathscr{G})$.

Only in the last three of these *rules*, conjuncts of the forms $\mathsf{G}\langle E\rangle\varphi$ and φ interact. For example, if $\mathscr{A} \rightarrow \mathscr{G}$ holds along all traces of a computation tree, and we know that through every node in the tree, there is a trace on which $\mathscr{A}$ holds, then along this trace, $\mathscr{A} \wedge \mathscr{G}$ holds as well. Thus, we also know that the computation tree satisfies $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$. Note that $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$ is not equal to $\mathsf{G}\langle E\rangle(\mathscr{G}) \wedge \mathsf{G}\langle E\rangle(\mathscr{A})$ because there exist computation trees for which a part of their traces satisfy $\mathscr{A} \wedge \neg\mathscr{G}$, the other traces satisfy $\mathscr{G} \wedge \neg\mathscr{A}$, but none of their traces satisfy $\mathscr{A} \wedge \mathscr{G}$.

After reducing the set of distinct cooperation levels by these reduction rules, we obtain only 15 semantically different subsets of D, which form a lattice with its partial order defined by implication. We leave out the true element of the lattice in the following (as it is trivially satisfied by all computation trees), and visualize the remaining cooperation levels hierarchically in Fig. 1. For every cooperation level, implied elements of D have also been left out in the vertex labeling. Every edge in the figure denotes a "stricter than"-relation, with the upper element being the stricter one. Vertices that enforce the traditional correctness criterion $\mathscr{A} \rightarrow \mathscr{G}$ are colored in gray.

In the next section, we discuss the different levels of the cooperation hierarchy in an example. After that, we will discuss design decisions that we made when constructing the hierarchy, as well as extensions and alternatives.

4.2 Example

This section presents an example from real life to illustrate some cooperation levels from Fig. 1. A more extensive example will be presented in Appendix A.

An IT provider makes a contract with a customer. The provider promises that whenever the computer breaks, it will eventually deliver a new computer, or one of its technicians will come to the customer to fix the computer while providing some free printing supplies as a bonus. The customer accepts the conditions, mainly because of the possibility of free printing supplies. The contract between the customer and the IT provider contains the following assumptions under which it has to be fullfilled: (1) every traffic blockade on the way to the customer is eventually cleared, and (2) if the road to the customer is clear and the IT provider technician is working on the computer, then it will eventually be fixed. The latter assumption is part of the contract to exclude problems with the customer's software, which is not the responsibility of the IT provider. By simply replacing parts of the computer one-by-one, the problem is eventually fixed. We also have an additional assumption that traffic can be blocked on purpose by the IT provider, which is not written into the contract.

We model the setting with two Boolean input variables c, b and four Boolean output variables t, d, m, n for the IT service provider's behavior. Variable c is true if the computer is currently working, b is true when the traffic is not blocked, $f = $ true indicates that the IT provider's technician is trying to fix the computer, $m = $ true means that free printing supplies are delivered, $n = $ true means that the IT service provider delivers a new computer, and $d = $ true means that the provider currently blocks the traffic on purpose. In LTL syntax[1], the guarantee can be written as $\mathscr{G} = G(\neg c \rightarrow F(n \vee (c \wedge m)))$. The assumptions can be formalized as $\mathscr{A} = G(Fb) \wedge G((b \wedge f) \rightarrow Fc) \wedge G(d \rightarrow \neg b)$.

The following table summarizes some cooperation levels that can be achieved with different behavior of the IT provider, each expressed as an LTL property whose fulfillment completely determines the valuation of the output variables in all time steps. We focus on levels that enforce $\mathscr{A} \rightarrow \mathscr{G}$ (which are colored gray in Fig. 1).

Nr.	Behavior	Level in Fig. 1
1	$G(d \wedge \neg f \wedge \neg m \wedge \neg n)$	$\mathscr{A} \rightarrow \mathscr{G}$
2	$G(d \wedge m \wedge \neg f \wedge \neg n)$	$(\mathscr{A} \rightarrow \mathscr{G}) \wedge G\langle E \rangle \mathscr{G}$
3	$G(f \wedge m \wedge \neg n \wedge \neg d)$	$(\mathscr{A} \rightarrow \mathscr{G}) \wedge G\langle E \rangle \mathscr{A}$
4	$G(n \wedge d \wedge \neg f \wedge \neg m)$	$\mathscr{G}$
5	$G(n \wedge \neg d \wedge \neg f \wedge \neg m)$	$\mathscr{G} \wedge G\langle E \rangle \mathscr{A}$

Behavior 1 enforces an assumption violation by blocking the traffic. This is very uncooperative with the customers (and all other drivers on the streets). In this case, $G\langle E \rangle \mathscr{A}$ does not hold as the environment cannot satisfy $\mathscr{A}$ from any point in any trace because it cannot set b to true at some point without violating $\mathscr{A}$, but not ever doing so violates $\mathscr{A}$ as well. The cooperation level specification part $G\langle E \rangle \mathscr{G}$ does not hold either because since n and m are both false all of the time, $\mathscr{G}$ is not fulfilled along any trace.

Behavior 2 is better because printing supplies are always delivered. This satisfies $G\langle E \rangle \mathscr{G}$ because, at any point, the environment can make the computer work again (which occasionally happens with computers). $G\langle E \rangle \mathscr{A}$ is still not satisfied for the same reason as before.

---

[1]In LTL syntax, G means "always" and F means "eventually".

Behavior 3 is even less destructive. The technician tries to fix the computer ($f = $ true) and brings along free printing supplies ($m = $ true) without blocking traffic. This satisfies the guarantees if the assumptions are satisfied. On top of that, the assumptions can always be satisfied by setting both b and c to true. Since d is always false, the last assumption (which was the problem before) always holds.

Behavior 4 is better than Behavior 2 but incomparable with Behavior 3. By always (or repeatedly) providing a new computer, $\mathscr{G}$ is enforced independent of the assumptions. However, $\mathsf{G}\langle E\rangle\mathscr{A}$ does not hold because, since the traffic is blocked ($d = $ true), the assumptions cannot hold.

Behavior 5 is similar but without enforcing a traffic blockade. It thus satisfies $\mathscr{G}$ and $\mathsf{G}\langle E\rangle\mathscr{A}$ simultaneously. The last two of the four assumptions are even enforced. However, since the first assumption cannot be enforced by any behavior of the IT provider, $\mathscr{G}\wedge\mathscr{A}$ cannot be achieved.

4.3   Discussion

Before solving the *cooperative synthesis* problem in the next section, let us discuss some interesting aspects of our hierarchy of cooperation levels.

Incomparable levels: As already discussed in the example, our hierarchy of cooperation levels in Fig. 1 contains levels that are incomparable, i.e., where neither of the levels is stricter or clearly more desirable. The relative preferences between such levels may depend on the application. Among the levels that enforce $\mathscr{A}\rightarrow\mathscr{G}$ (gray in Fig. 1), there is only one incomparability, namely between $\mathscr{G}$ and $(\mathscr{A}\rightarrow\mathscr{G})\wedge$ $\mathsf{G}\langle E\rangle\mathscr{A}$. The former favors the guarantees, the latter the assumptions. For incomparabilities between a level that enforces $\mathscr{A}\rightarrow\mathscr{G}$ and one that does not, we suggest to prefer the former unless there are good reasons not to.

Symmetry in Fig. 1: Our hierarchy is asymmetric because we included $\mathscr{A}\rightarrow\mathscr{G}$ but not $\mathscr{G}\rightarrow\mathscr{A}$ in D. Including $\mathscr{A}\rightarrow\mathscr{G}$ contradicts the philosophy of cooperation to some extend, but is justified by the fact that we always take the point of view of the system in this paper, and try to synthesize a *correct implementation* that also helps the environment to satisfy its assumptions whenever this is reasonable, but without assuming that the environment has its own goals and behaves *rationally* (as in [3,10–12,18]). The guarantees often cannot be enforced unconditionally by the system, so the system has to rely on the assumptions to hold. However, the combination of $\mathscr{A}\rightarrow\mathscr{G}$ with $\mathsf{G}\langle E\rangle\mathscr{A}$ (or $\mathsf{G}\langle E\rangle\mathscr{G}$) in a cooperation level specification eliminates the possibility for the system to achieve correctness by simply enforcing a violation of $\mathscr{A}$ (and $\mathscr{G}$).

Additional operators: We did not include any properties with a plain $\langle E\rangle$-operator in our default hierarchy. The reason is that $\langle E\rangle\,\varphi$ is quite a weak goal. The system could have exactly one trace on which φ is satisfied. As soon as the environment deviates from the input along this trace, the system can behave arbitrarily. Our default hierarchy also does not contain any occurrences of $\mathscr{A}\vee\mathscr{G}$, but this is mainly to keep the presentation simple. Including $\mathscr{A}\vee\mathscr{G}$ and $\mathsf{G}\langle E\rangle(\mathscr{A}\vee\mathscr{G})$ would extend the hierarchy to 23 levels. Additionally including $\langle E\rangle$ wherever $\mathsf{G}\langle E\rangle$ is applied results in 77 levels. Thus, even with extensions in the applied operators, the size of our hierarchy remains manageable. Details to these refined hierarchies and the reduction rules to obtain them can be found in Appendix B.

Fine-grainedness: Our hierarchy considers two dimensions: the goals (in terms of assumptions and guarantees) to achieve, and the certainty with which the goal can be achieved: enforced, always reachable ($\mathsf{G}\langle E\rangle$), and initially reachable ($\langle E\rangle$, if considered). In contrast, most existing synthesis approaches [4,5, 16] (see Section 2) that go beyond plain correctness only focus on *enforcing* a maximum in the dimension of goals. Yet, in this dimension they are often more fine-grained, e.g., by considering individual assumptions and guarantees or how often some property is violated. In principle, we could also increase

the granularity in our goal dimension. However, this also comes at a price: it would increase the size of the hierarchy and induce more incomparabilities, which makes it more difficult to define the preference between the incomparable levels for a concrete application.

5 Synthesizing Desirable Systems

After defining our hierarchy of cooperation levels, we turn towards the synthesis of implementations that maximize the possible cooperation level. We start by describing how we can synthesize an implementation for a single cooperation level, and then show in Sect. 5.2 how to synthesize *maximally cooperative implementations*, which move upwards in the cooperation level hierarchy whenever possible during the execution.

5.1 Implementing a Single Cooperation Level

The simple reduction of the cooperative synthesis problem for LTL to synthesis from a logic such as *strategy logic* [13] is obstructed by the fact that in our semantics, we always evaluate traces from the start when evaluating a subformula of the shape $\mathsf{G}\langle E \rangle \varphi$. Strategy logic lacks a *rewind* operator that would allow to jump back to the start of a trace. We could, however, encode a cooperation level specification into CTL* with linear past [9], and use a corresponding synthesis procedure.

Instead of following this route, we give a direct automata-theoretic approach to the synthesis of reactive systems that implement some level of cooperation with the environment on a specification $(\mathscr{A}, \mathscr{G})$. The automata built in this way keep information about the states of the assumptions and guarantees explicit, which is needed to synthesize maximally cooperative implementations in the next subsection. Also, it makes the synthesis approach applicable to general ω-regular word language specifications.

Starting from the linear-time properties $\mathscr{A}$ and $\mathscr{G}$, we show how to build a non-deterministic Rabin tree automaton that encodes the synthesis problem for some cooperation level specification H. Such a tree automaton can be checked for language emptiness in order to perform synthesis for one level of cooperation.

Let $\mathscr{R}_{\mathscr{A}}$, $\mathscr{R}_{\mathscr{G}}$, $\mathscr{R}_{\mathscr{A} \to \mathscr{G}}$, and $\mathscr{R}_{\mathscr{A} \land \mathscr{G}}$ be deterministic Rabin word automata that encode $\mathscr{A}$, $\mathscr{G}$, $\mathscr{A} \to \mathscr{G}$, and $\mathscr{A} \land \mathscr{G}$. We translate the conjuncts (elements of D from Equation 2) of some expression H in the grammar from Equation 1 to non-deterministic Rabin tree automata $\mathscr{T}^D$ individually and then build the product $\mathscr{T}^H$ between these tree automata, which encodes that all elements of H have to be satisfied in a candidate computation tree. As cooperation level specifications have only three types of conjuncts, namely φ, $\langle E \rangle \varphi$, and $\mathsf{G}\langle E \rangle \varphi$ for some linear-time property φ, we simply give the translations for these types separately.

Case φ: If φ is a linear-time property, the occurrence of φ in a cooperation level specification indicates that φ should hold on every trace of a synthesized implementation. Translating $\mathscr{R}_{\varphi} = (Q, \mathscr{I} \times \mathscr{O}, \delta, q_0, \mathscr{F})$ to a Rabin tree automaton $\mathscr{T}^D = (Q, \mathscr{I}, \mathscr{O}, \delta', q_0, \mathscr{F})$ that enforces φ to hold along all traces in the tree is a standard construction, where we set:

$$\forall q \in Q, o \in \mathscr{O} : \delta'(q, o) = \{\{i \mapsto \delta(q, (i, o)) \mid i \in \mathscr{I}\}\}$$

Case $\langle E \rangle \varphi$: Here, we require the synthesized system to offer one path along which φ holds. In the tree automaton, we non-deterministically choose this path. Starting with $\mathscr{R}_{\varphi} = (Q, \mathscr{I} \times \mathscr{O}, \delta, q_0, \mathscr{F})$, we

obtain $\mathscr{T}^D = (Q, \mathscr{I}, \mathscr{O}, \delta', q_0, \mathscr{F})$ with:

$$\forall q \in Q, o \in \mathscr{O} : \delta'(q, o) = \bigcup_{i \in \mathscr{I}} \{\{i \mapsto \delta(q, (i, o))\} \cup \{i' \mapsto \top \mid i' \in \mathscr{I}, i' \neq i\}\}$$

Case $\mathsf{G}\langle E\rangle\varphi$: In this case, for every node in a computation tree, we require the synthesized system to have a path on which φ holds that includes the selected node. Given $\mathscr{R}_\varphi = (Q, \mathscr{I} \times \mathscr{O}, \delta, q_0, \mathscr{F})$, we can implement this requirement as a non-deterministic Rabin tree automaton $\mathscr{T}^D = (Q', \mathscr{I}, \mathscr{O}, \delta', q_0', \mathscr{F}')$ with:

$$Q' = Q \times \mathbb{B}$$

$$\forall (q, b) \in Q', o \in \mathscr{O} : \delta'((q, b), o) = \bigcup_{i \in \mathscr{I}} \{\{i \mapsto (\delta(q, (i, o)), \mathsf{true})\}$$
$$\cup \{i' \mapsto (\delta(q, (i', o)), \mathsf{false}) \mid i' \in \mathscr{I}, i' \neq i\}\}$$

$$q_0' = (q_0, \mathsf{true})$$

$$\mathscr{F}' = \{(F \times \{\mathsf{true}\}, G \times \{\mathsf{true}\}) \mid (F, G) \in \mathscr{F}\}$$
$$\cup \{(\emptyset, Q \times \{\mathsf{false}\})\}$$

The automaton augments the states in Q by a Boolean flag. From every node in a computation tree accepted by $\mathscr{T}^D$, regardless of whether it is flagged by true or false, there must exist a branch consisting only of true-labeled nodes. The original acceptance condition $\mathscr{F}$ must hold along this branch. However, not all branches of a tree accepted by $\mathscr{T}^D$ have to satisfy φ, as those branches along which the flag is false infinitely often are trivially accepting. Intuitively, $\mathscr{T}^D$ also enforces the safety hull of φ along all branches in the computation tree as once a bad prefix has been seen on the way to a computation tree node, there cannot exist a branch containing the node along which φ holds, which is enforced by the (required) successor branch that is always labeled by true.

In order to obtain a non-deterministic Rabin tree automaton $\mathscr{T}^H$ for a complete cooperation level specification H, we can compute a product automaton from the Rabin tree automata $\mathscr{T}^D$ for the individual conjuncts. Computing such a product automaton is a standard operation in automata theory. Given a set $\{(Q_1, \mathscr{I}, \mathscr{O}, \delta_1, q_{0,1}, \mathscr{F}_1), \ldots, (Q_n, \mathscr{I}, \mathscr{O}, \delta_n, q_{0,n}, \mathscr{F}_n)\}$ of non-deterministic Rabin tree automata, their product is defined as the non-deterministic Rabin tree automaton $\mathscr{T}^H = (Q, \mathscr{I}, \mathscr{O}, \delta, q_0, \mathscr{F})$ with:

$$Q = Q_1 \times \ldots \times Q_n$$

$$\delta((q_1, \ldots, q_n), o) = \bigotimes_{i \in \{1, \ldots, n\}} \delta_i(q_i, o)$$

$$q_0 = (q_{0,1}, \ldots, q_{0,n})$$

$$\mathscr{F} = \{(F_1 \times \ldots \times F_n, G_1 \times \ldots \times G_n) \mid$$
$$(F_1, G_1) \in \mathscr{F}_1, \ldots, (F_n, G_n) \in \mathscr{F}_n\}$$

The second line of this equation holds for all $(q_1, \ldots, q_n) \in Q$ and $o \in \mathscr{O}$. Also, we used the special operator $\bigotimes$ that maps sets $\{M_i \subseteq 2^{Q_i \to \mathscr{I}}\}_{j \in \{1, \ldots, n\}}$ to the set

$$\{\{((q_1, \ldots, q_n), i) \in Q \times \mathscr{I} \mid \bigwedge_{j \in \{1, \ldots, n\}} f_j(i) = q_j\} \mid f_1 \in M_1, \ldots, f_n \in M_n\}$$

By performing reactive synthesis using the product Rabin tree automaton as specification, we can obtain an implementation that falls into the chosen cooperation level.

5.2 Maximally Cooperative Synthesis

For linear-time specifications $\mathscr{A}$ and $\mathscr{G}$, there are typically some cooperation level specifications that are not realizable (such as $\mathscr{A} \wedge \mathscr{G}$) and some that are realizable (such as $\mathsf{G}\langle E\rangle(\mathscr{A} \to \mathscr{G})$). By iterating over all cooperation level specifications and applying reactive synthesis for the product Rabin tree automata $\mathscr{T}'$ computed by the construction from above, we can check which is the highest cooperation level that can be realized and compute a respective implementation.

In some cases, there may however be the possibility to switch to a higher level of cooperation during the system's execution. Take for example the case that initially, the cooperation level $\mathsf{G}\langle E\rangle\mathscr{A} \wedge (\mathscr{A} \to \mathscr{G})$ is not realizable, but $(\mathscr{A} \to \mathscr{G})$ is. This may be the case if $\mathscr{A}$ represents the constraint that the environment has to perform a certain finite sequence of actions in reaction to the system's output, which is representable as a safety property. If the actions are triggered by the system, then the system cannot ensure that the environment succeeds with performing them correctly, hence violating $\mathscr{A}$. If $\mathscr{A} \to \mathscr{G}$ can only be realized by the system by triggering these actions at least once, then after these actions have been performed by the environment, the cooperation level specification $\mathsf{G}\langle E\rangle\mathscr{A} \wedge (\mathscr{A} \to \mathscr{G})$ can however be enforced by the system by not triggering them again.

This observation motivates the search for *maximally cooperative implementations*, which at any point in time realize the highest possible cooperation level. Before describing how to synthesize such implementations, let us first formally define what this means.

When determining the cooperation level during the execution of a system, we only look at the part of its computation tree that is consistent with the input obtained from the environment so far. Given a computation tree $\langle T, \tau\rangle$ and the input part of a prefix trace $w^{\mathscr{I}} = w^{\mathscr{I}}_0 w^{\mathscr{I}}_1 \ldots w^{\mathscr{I}}_n \in \mathscr{I}^*$, we define the *bobble tree* [14] of $\langle T, \tau\rangle$ for $w^{\mathscr{I}}$ to be the tree $\langle T', \tau'\rangle$, where $T' = \{\varepsilon\} \cup \{w^{\mathscr{I}}_0 w^{\mathscr{I}}_1 \ldots w^{\mathscr{I}}_k \mid k \le n\} \cup \{w^{\mathscr{I}} t \mid t \in \mathscr{I}^*\}$, and $\tau'(t) = \tau(t)$ for all $t \in T'$. We call $w^{\mathscr{I}}$ the *split node* of $\langle T', \tau'\rangle$ and $(\tau(\varepsilon), w^{\mathscr{I}}_0)(\tau(w^{\mathscr{I}}_0), w^{\mathscr{I}}_1) \ldots (\tau(w^{\mathscr{I}}_0 \ldots w^{\mathscr{I}}_{n-1}), w^{\mathscr{I}}_n)$ the *split word* of $\langle T', \tau'\rangle$. Intuitively, the bobble tree has a single path to the split node $w^{\mathscr{I}}$ and is full from that point onwards. Cutting a full computation tree into a bobble tree does not reduce the cooperation level that the tree fulfills for the specification types in the classes that we can built from the conjuncts in D (from Eqn. 2):

Lemma 1. *Let $\langle T, \tau\rangle$ be a computation tree and $\langle T', \tau'\rangle$ be a bobble tree built from $\langle T, \tau\rangle$. If for some cooperation level specification H consisting of conjuncts in D, we have that $\langle T, \tau\rangle$ fulfills H, then $\langle T', \tau'\rangle$ also fulfills H.*

Proof. Proof by induction over the structure of H, using the semantics given on page 5. All conjuncts in D have an outermost universal quantification over the elements in $L(\langle T, \tau\rangle)$. Reducing the number of elements in $L(\langle T, \tau\rangle)$ does not make these constraints harder to fulfill. $\qquad\square$

Bobble trees provide us with a semantical basis for switching between cooperation levels: if a reactive system $\langle T, \tau\rangle$ for a cooperation level H executes, and after some prefix trace w, there exists a bobble tree with split word w that allows a strictly higher cooperation level H', then it makes sense to continue the execution of the system according to cooperation level H'. We thus define:

Definition 1. *Let $(\mathscr{A}, \mathscr{G})$ be a linear-time specification. We call a computation tree $\langle T, \tau\rangle$ maximally cooperative if for every split node $t \in T$, the bobble tree induced by t and $\langle T, \tau\rangle$ implements a highest possible cooperation level for $\mathscr{A}$ and $\mathscr{G}$ among the bobble trees with the same split word.*

Note that since for some bobble tree, there may be multiple highest cooperation levels, it makes sense to define a preference order for the cooperation level specifications, so that when synthesizing an

implementation, the implementation can always pick the most desired one when the possibility to move up in the hierarchy arises.

In order to synthesize maximally cooperative implementations, we first need a way to check, for every split word, for the existence of a bobble tree for cooperation level specifications. The special structure of the tree automata $\mathscr{T}^H$ built according to the construction in Sect. 5.1 offers such a way.

Definition 2. *Let $\mathscr{T}^H = (Q, \mathscr{I}, \mathscr{O}, \delta, q_0, \mathscr{F})$ be a non-deterministic tree automaton for a cooperation level specification built according to the (product) construction from Sect. 5.1. We have that Q is of the shape $C_1 \times \ldots \times C_n$, where for every $i \in \{1, \ldots, n\}$, we either have $C_i = Q'$ or $C_i = Q' \times \mathbb{B}$ for some Rabin word automaton state set Q'. For a state $q = (c_1, \ldots, c_n) \in Q$, we define $unpack(q) = unpack(c_1) \cup \ldots \cup unpack(c_n)$, where we concretize $unpack(q') = q'$ and $unpack((q', b)) = q'$ for some word automaton state $q' \in Q'$ and $b \in \mathbb{B}$.*

Lemma 2. *Let $\mathscr{T}^H$ be a tree automaton built according to the (product) construction from Sect. 5.1 from a cooperation level specification with conjuncts in D over the linear-time specifications $\mathscr{A}$, $\mathscr{G}$, $\mathscr{A} \to \mathscr{G}$, and $\mathscr{A} \wedge \mathscr{G}$. Let those linear-time specifications be represented by Rabin word automata with the state sets $Q_{\mathscr{A}}$, $Q_{\mathscr{G}}$, $Q_{\mathscr{A} \to \mathscr{G}}$, and $Q_{\mathscr{A} \wedge \mathscr{G}}$, respectively. Without loss of generality, let these state sets be disjoint. We have that:*

1. *For all reachable states q in $\mathscr{T}^H$, there is at most one state in $unpack(q)$ from each of $Q_{\mathscr{A}}$, $Q_{\mathscr{G}}$, $Q_{\mathscr{A} \to \mathscr{G}}$, and $Q_{\mathscr{A} \wedge \mathscr{G}}$.*

2. *All states in $\mathscr{T}^H$ with the same set $unpack(q)$ have the same languages.*

Proof. The first claim follows directly from the constructions from Sec. 5.1: for every state in the individual Rabin tree automata $\mathscr{T}^D$ built from cooperation level specification conjuncts of the shapes $\mathsf{G}\langle E \rangle \varphi$ and φ, the automata always track the state of the corresponding word automata for a branch of the tree.

For the second claim, we decompose the states in $\mathscr{T}^H$ into their factors and prove the claim for each factor individually. For factors originating from cooperation level specifications of the shape φ for some linear-time property φ, this fact is trivial. For factors originating from specification conjuncts of the shape $\mathsf{G}\langle E \rangle \varphi$, the claim follows from the fact that the tree automaton states that only differ in their Boolean flag have the same successor state functions. $\qquad\square$

Lemma 2 tells us how we can switch between cooperation levels. Assume that we can always read off *all* current states of $Q_{\mathscr{A}}$, $Q_{\mathscr{G}}$, $Q_{\mathscr{A} \to \mathscr{G}}$, and $Q_{\mathscr{A} \wedge \mathscr{G}}$ from *unpack(q)* for any state q of a product tree automaton $\mathscr{T}^H$. This assumption can be made satisfied by letting $\mathscr{T}^H$ be the product of *all* elements in the set of considered cooperation level specification conjuncts D, but only using the ones in the current cooperation level H when building the acceptance condition of $\mathscr{T}^H$. Now consider a second cooperation level specification H' that is higher in the hierarchy than H and its associated tree automaton $\mathscr{T}^{H'}$. Let W be the states in $\mathscr{T}^H$ with a non-empty language and W' be the states of $\mathscr{T}^{H'}$ with a non-empty language. If we find a state q' in $\mathscr{T}^{H'}$ for which $unpack(q') = unpack(q)$, and state q' has a non-empty language, then we can simply re-route every transition to q to q' and obtain a new tree automaton with the states in $\mathscr{T}^H$ and $\mathscr{T}^{H'}$ that enforces a higher cooperation level on a bobble tree along all branches in run trees that lead to q. If we now identify the non-empty tree automaton states for all cooperation level specifications in our hierarchy, and apply this approach to all pairs of the corresponding tree automata and all of their states, we end up with a tree automaton that accepts maximally cooperative computation trees. More formally, this line of reasoning shows the correctness of the following construction:

Definition 3. *Let $H_1, \ldots, H_{14}$ be the cooperation level specifications of our hierarchy, ordered by preference and respecting the hierarchy's partial order $\leq_H$, and let $\mathscr{T}_1^H, \ldots, \mathscr{T}_{14}^H$ be the non-deterministic*

Rabin tree automata for them. Let us furthermore rename states q in an automaton $\mathscr{T}_j^H$ to (q, j) to make their names unique, and let every tree automaton $\mathscr{T}_j^H$ be given as a tuple $(Q_j, \mathscr{I}, \mathscr{O}, \delta_j, q_{0,j}, \mathscr{F}_j)$. Let $W \subseteq \bigcup_j Q_j$ be the states in the tree automata with a non-empty language. We define the Rabin tree automaton $\mathscr{T} = (Q', \mathscr{I}, \mathscr{O}, \delta', q_0', \mathscr{F}')$ encoding the maximally cooperative synthesis problem as follows:

$$Q' = \bigcup_{j \in \{1,\dots,14\}} Q_j$$

$$q_0' = q_{0,j} \ for \ j = \max\{j \in \{1,\dots,14\} \mid q_{0,j} \in W\}$$

$$\mathscr{F}' = \mathscr{F}_1 \cup \dots \cup \mathscr{F}_{14}$$

$$\delta'((q,j),o) = \{\{i \mapsto (q'',j') \mid j' = \max\{k \in \{1,\dots,14\} \mid \exists (q'',k) \in Q_k :$$
$$q'' \in W, unpack(q'') = unpack(f(i)), H_j \leq_H H_k\},$$
$$unpack(q'') = unpack(f(i)),$$
$$((j = j') \to q'' = f(i))\} \mid f \in \delta_j(q,o)\}$$
$$for \ all \ (q,j) \in Q' \ and \ i \in \mathscr{I}$$

Theorem 1. *A Rabin tree automaton built from linear-time specifications $\mathscr{A}$ and $\mathscr{G}$ according to Def. 3 encodes the maximally cooperative synthesis problem for the specification $(\mathscr{A}, \mathscr{G})$. Building the tree automaton and checking it for emptiness can be performed in doubly-exponential time for specifications in LTL.*
For specifications given as deterministic Rabin word automata, the time complexity is polynomial in the number of states and exponential in the number of acceptance pairs.

Proof. For the correctness, note that the tree automaton can switch between cooperation levels only finitely often, and whenever it switches, it only does so to strictly higher levels of cooperation.

To obtain doubly-exponential time complexity of maximally cooperative synthesis from LTL specifications, we first translate $\mathscr{A}$, $\mathscr{G}$, $\mathscr{A} \to \mathscr{G}$, and $\mathscr{A} \wedge \mathscr{G}$ to deterministic Rabin word automata with a doubly-exponential number of states and a singly-exponential number of acceptance condition pairs, which takes doubly-exponential time. The overall sizes of the tree automata build for the cooperation levels are then polynomial in the sizes of the Rabin word automata. We can compute W in time exponential in the number of acceptance pairs and polynomial in the number of tree automaton states, which sums up to doubly-exponential time (in the lengths of $\mathscr{A}$ and $\mathscr{G}$) for LTL. When building $\mathscr{T}$ and computing the accepted computation tree, the same argument applies. If the specification is given in form of deterministic Rabin word automata, then all the product automata computed in the process have a number of states that is polynomial in the number of states of the input automata and a number of acceptance pairs that is polynomial in the number of acceptance pairs of the input automata. By the complexity of checking Rabin tree automata for emptiness and computing W, the second claim follows as well.

$\square$

Theorem 1 states that synthesizing maximally cooperative implementations from LTL specifications does not have a higher complexity than LTL synthesis in general. Note, however, that both the synthesis time and the size of the resulting systems can increase in practice as the Rabin automata built in cooperative synthesis are larger than in standard LTL synthesis.

Also note that we can extend the theory from this subsection to also include cooperation level specification conjuncts of the shape $\langle E \rangle \varphi$. However, we would need to add flags to the tree automata to keep track of whether the current branch in a computation tree is the one on which φ should hold. As

these flags need to be tracked along changes between the cooperation levels, the definitions from this subsection would become substantially more complicated. Thus, we refrained from doing so here.

6   Conclusion

Conventional synthesis algorithms often produce systems that actively work towards violating environment assumptions rather than satisfying assumptions and guarantees together. In this paper, we worked out a fine-grained hierarchy of cooperation levels between the system and the environment for satisfying both guarantees *and* assumptions as far as possible. We also presented a synthesis procedure that maximizes the cooperation level in the hierarchy for linear-time specifications, such as Linear Temporal Logic (LTL). The worst-case complexity of this procedure for LTL is the same as of conventional LTL synthesis. Our approach relieves the user from requiring cooperation in the specification explicitly, which helps to keep the specification clean and abstract.

In the future, we plan to work out cooperative synthesis procedures for other specification languages, and evaluate the results on industrial applications.

References

[1] S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *ICALP*, pages 15–27. Springer, 2013.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[3] D. Berwanger. Admissibility in infinite games. In *STACS*, pages 188–199, 2007.

[4] R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, G. Hofferek, B. Jobstmann, B. Könighofer, and R. Könighofer. Synthesizing robust systems. *Acta Inf.*, 51(3-4):193–220, 2014.

[5] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156, 2009.

[6] R. Bloem, R. Ehlers, S. Jacobs, and R. Könighofer. How to handle assumptions in synthesis. In *SYNT*, pages 34–50, 2014.

[7] R. Bloem, R. Ehlers, and R. Könighofer. Cooperative reactive synthesis. In *ATVA*. Springer, 2015. To appear.

[8] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[9] Laura Bozzelli. The complexity of CTL* + linear past. In *Foundations of Software Science and Computational Structures (FOSSACS)*, pages 186–200, 2008.

[10] R. Brenguier, J.-F. Raskin, and M. Sassolas. The complexity of admissibility in omega-regular games. In *CSL-LICS*, page 23, 2014.

[11] K. Chatterjee, L. Doyen, E. Filiot, and J.-F. Raskin. Doomsday equilibria for omega-regular games. In *VMCAI*, pages 78–97, 2014.

[12] K. Chatterjee and T. A. Henzinger. Assume-guarantee synthesis. In *TACAS*, 2007.

[13] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

[14] R. Ehlers and B. Finkbeiner. Monitoring realizability. In *RV*, pages 427–441, 2011.

[15] R. Ehlers, R. Könighofer, and R. Bloem. Synthesizing cooperative reactive mission plans. In *IROS*. IEEE, 2015.

[16] R. Ehlers and U. Topcu. Resilience to intermittent assumption violations in reactive synthesis. In *HSCC*, pages 203–212, 2014.

[17] M. Faella. Admissible strategies in infinite games over graphs. In *MFCS*, 2009.

[18] D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *TACAS*, 2010.

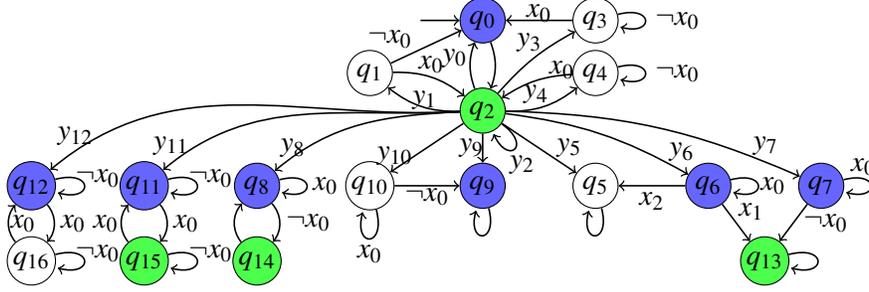[19] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

Figure 2: Example specification to illustrate all cooperation levels from Fig. 1.

A   Another Example

To complement Section 4.2, this section illustrates *all* cooperation levels from Fig. 1 on a more technical example. The assumptions $\mathscr{A}$ are defined using the Rabin word automaton $\mathscr{R}_{\mathscr{A}} = (Q, \mathscr{I} \times \mathscr{O}, \delta, q_0, \{(\emptyset, G_{\mathscr{A}})\})$ shown in Fig. 2, where $Q = \{q_0, \ldots, q_{16}\}$, the input alphabet is $\mathscr{I} = \{x_0, x_1, x_2\}$, and the output alphabet is $\mathscr{O} = \{y_0, \ldots, y_{12}\}$. Fig. 2 labels edges with conditions over the input and output letters to define the transition function δ. For instance, the condition $\neg x_0$ means that the transition is taken for all letters $(x, y) \in \mathscr{I} \times \mathscr{O}$ where $x \neq x_0$. Edges without a label are always taken. Note that the system can only influence the next state from q_2: with output y_i, the next state will be q_i. The acceptance condition is defined with $G_{\mathscr{A}} = \{q_0, q_6, q_7, q_8, q_9, q_{11}, q_{12}\}$. That is, the blue states in Fig. 2 must be visited infinitely often for $\mathscr{A}$ to be satisfied. The guarantees $\mathscr{G}$ are defined using the automaton $\mathscr{R}_{\mathscr{G}} = (Q, \mathscr{I} \times \mathscr{O}, \delta, q_0, \{(\emptyset, G_{\mathscr{G}})\})$, which differs from $\mathscr{R}_{\mathscr{A}}$ only in the acceptance condition: with $G_{\mathscr{G}} = \{q_2, q_{13}, q_{14}, q_{15}\}$, $\mathscr{G}$ is satisfied if the green states in Fig. 2 are visited infinitely often.

The following paragraphs present and discuss one system behavior (defined in the form of a computation tree) per cooperation level in Fig. 1.

$\underline{\mathscr{A} \to \mathscr{G}}$: The computation tree $\langle \mathscr{I}^*, \tau_5 \rangle$ with $\tau_5(w^{\mathscr{I}}) = y_5$ for all $w^{\mathscr{I}} \in \mathscr{I}^*$ represents a system that always outputs y_5. It satisfies $\mathscr{A} \to \mathscr{G}$ because q_5 is a trap where neither $\mathscr{A}$ nor $\mathscr{G}$ is satisfied. This is "correct" but rather unsatisfactory.

$\underline{(\mathscr{A} \to \mathscr{G}) \wedge \mathsf{G}\langle E \rangle \mathscr{G}}$: is satisfied by the computation tree $\langle \mathscr{I}^*, \tau_4 \rangle$ with $\tau_4(w^{\mathscr{I}}) = y_4$. Again, $\mathscr{A} \to \mathscr{G}$ holds because staying in $\{q_2, q_4\}$ violates $\mathscr{A}$. Still, at any point in time, there exists some future environment behavior that satisfies $\mathscr{G}$ (namely one that choses x_0 infinitely often). Thus, τ_4 is slightly better than τ_5.

$\underline{(\mathscr{A} \to \mathscr{G}) \wedge \mathsf{G}\langle E \rangle \mathscr{A}}$: is satisfied by $\langle \mathscr{I}^*, \tau_3 \rangle$ with $\tau_3(w^{\mathscr{I}}) = y_3$: At any point in time, $\mathscr{A}$ can be satisfied for some future environment behavior (namely if x_0 is chosen in q_3 infinitely often). Since $\mathscr{A} \to \mathscr{G}$ (because of the unconditional edge from q_0 to q_2), this implies that also $\mathscr{G}$ can be reached. Yet, neither $\mathscr{A}$ nor $\mathscr{G}$ are enforced because the environment could always give x_1 to get stuck in q_3. However, assuming that the environment does not behave totally self-destructive, both $\mathscr{A}$ and $\mathscr{G}$ will be satisfied. Thus, τ_3 is even better than τ_4.

$\underline{\mathscr{G}}$: The computation tree with $\tau_2(w^{\mathscr{I}}) = y_2$ enforces $\mathscr{G}$, but defeats any hope of reaching $\mathscr{A}$. It is thus better than τ_2 but incomparable with τ_3: Enforcing $\mathscr{G}$ is better than having $\mathscr{G}$ reachable, but $\mathscr{A}$ being unreachable is worse than having $\mathscr{A}$ reachable. Hence, the choice between τ_2 and τ_3 is a question of selfishness.

$\underline{\mathscr{G} \wedge \mathsf{G}\langle E \rangle \mathscr{A}}$: is satisfied by $\tau_1(w^{\mathscr{I}}) = y_1$: No matter if the environment picks x_0 or not in q_1, the state q_2 will always be visited because of the unconditional edge from q_0 to q_2. Visiting q_0 infinitely often

(and thus satisfying $\mathscr{A}$) is always possible, but not enforced. Hence, τ_1 dominates both τ_2 and τ_3.

$\underline{\mathscr{G} \wedge \mathscr{A}}$: is enforced by $\tau_0(w^{\mathscr{I}}) = y_0$: independent of the environment, both q_0 and q_2 are visited repeatedly. Thus, τ_0 is the most desirable system behavior.

The remaining cooperation levels from the hierarchy in Fig. 1 do not satisfy the traditional correctness criterion $\mathscr{A} \to \mathscr{G}$. Still, a computation tree for such a level behaves better than arbitrarily in various ways, and are useful from states where $\mathscr{A} \to \mathscr{G}$ cannot be enforced.

$\underline{\mathsf{G}\langle E\rangle(\mathscr{A} \to \mathscr{G})}$: is satisfied by the computation tree $\langle \mathscr{I}^*, \tau_6 \rangle$ with $\tau_6(w^{\mathscr{I}}) = y_6$ for all $w^{\mathscr{I}} \in \mathscr{I}^*$. At any point in time, there exists some environment behavior that either violates $\mathscr{A}$ (by going to q_5) or that satisfies $\mathscr{G}$ (by going to q_{13}). However, $\mathscr{G}$ alone is not reachable from every point in every execution: if the environment already traversed to q_5, $\mathscr{G}$ is lost. Similarly, $\mathscr{A}$ is not reachable at every point in time because the environment may already have traversed to q_{13}. Finally, $\mathscr{A} \to \mathscr{G}$ is not enforced because with input x_0 the environment may stay in q_6 forever, thereby satisfying $\mathscr{A}$ but not $\mathscr{G}$. Nevertheless, having $\mathscr{A} \to \mathscr{G}$ feasible at any point in time is better than nothing.

$\underline{\mathsf{G}\langle E\rangle \mathscr{G}}$: The computation tree with $\tau_7(w^{\mathscr{I}}) = y_7$ satisfies $\mathsf{G}\langle E\rangle \mathscr{G}$, and is thus better than τ_6. $\mathsf{G}\langle E\rangle \mathscr{A}$ does not hold because from q_{13}, $\mathscr{A}$ is unreachable. $\mathscr{A} \to \mathscr{G}$ is not enforced because the environment could stay in q_7 forever. Yet, τ_7 is still incomparable with τ_5 (enforcing $\mathscr{A} \to \mathscr{G}$), because with τ_5, there is not even a hope of reaching $\mathscr{G}$.

$\underline{\mathsf{G}\langle E\rangle \mathscr{A}}$: is similar to the previous case. The computation tree with $\tau_{10}(w^{\mathscr{I}}) = y_{10}$ satisfies $\mathsf{G}\langle E\rangle \mathscr{A}$, but $\mathscr{A}$ is not enforced because the environment can stay in q_{10}. Moreover, $\mathsf{G}\langle E\rangle(\mathscr{A} \to \mathscr{G})$ does not hold because it is unreachable from q_9.

$\underline{\mathscr{A}}$: The computation tree with $\tau_9(w^{\mathscr{I}}) = y_9$ enforces $\mathscr{A}$, and thus dominates τ_{10}. Yet all hopes for satisfying the guarantees are lost.

$\underline{\mathscr{A} \wedge \mathsf{G}\langle E\rangle \mathscr{G}}$: The system behavior $\tau_8(w^{\mathscr{I}}) = y_8$ enforces $\mathscr{A} \wedge \mathsf{G}\langle E\rangle \mathscr{G}$ and eliminates this defect of τ_9: if the environment is not hostile and produces x_1 or x_2 infinitely often, $\mathscr{G}$ is achievable.

$\underline{(\mathsf{G}\langle E\rangle \mathscr{A}) \wedge \mathsf{G}\langle E\rangle(\mathscr{A} \to \mathscr{G})}$: is satisfied by the system behavior $\tau_{12}(w^{\mathscr{I}}) = y_{12}$. The guarantee $\mathscr{G}$ is lost, but at any point in time, it is still possible that $\mathscr{A}$ is satisfied (by visiting s_{12} infinitely often) and that $\mathscr{A}$ is violated (by only visiting s_{16} from some point on). This is slightly better than τ_{10} (which satisfies $\mathsf{G}\langle E\rangle \mathscr{A}$ alone) because the correctness property $\mathscr{A} \to \mathscr{G}$ is reachable at any point. With τ_{10}, it may happen that the correctness property is lost once and for all.

$\underline{(\mathsf{G}\langle E\rangle \mathscr{A}) \wedge (\mathsf{G}\langle E\rangle \mathscr{G})}$ and $\underline{\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})}$: The system behavior $\tau_{11}(w^{\mathscr{I}}) = y_{11}$ satisfies $(\mathsf{G}\langle E\rangle \mathscr{A}) \wedge (\mathsf{G}\langle E\rangle \mathscr{G})$ as well as $\mathsf{G}\langle E\rangle(\mathscr{A} \wedge \mathscr{G})$. In our example, there is no way to distinguish these two levels because we chose objectives defined by visiting some states infinitely often. For other specification classes, there can be a difference, though. $\mathscr{A}$ is not enforced by τ_{11} because the environment could always stay in q_{15}. $\mathscr{A} \to \mathscr{G}$ is not enforced either because the environment could stick to q_{11}. Still, this is better than a prospect of reaching only $\mathscr{A}$ or only $\mathscr{G}$.

B   Extended Cooperation Hierarchies

As discussed in Section 4.3, we can extend our hierarchy in various ways. When extending the conjuncts from Eqn. 2 with $\mathscr{A} \vee \mathscr{G}$ and $\mathsf{G}\langle E\rangle(\mathscr{A} \vee \mathscr{G})$, we obtain the hierarchy shown in Figure 3, which has 23 levels. We applied the following reduction rules in addition to those mentioned in Section 4.1:

- $\mathscr{G}$ implies $\mathscr{A} \vee \mathscr{G}$,

- $\mathscr{A}$ implies $\mathscr{A} \vee \mathscr{G}$,

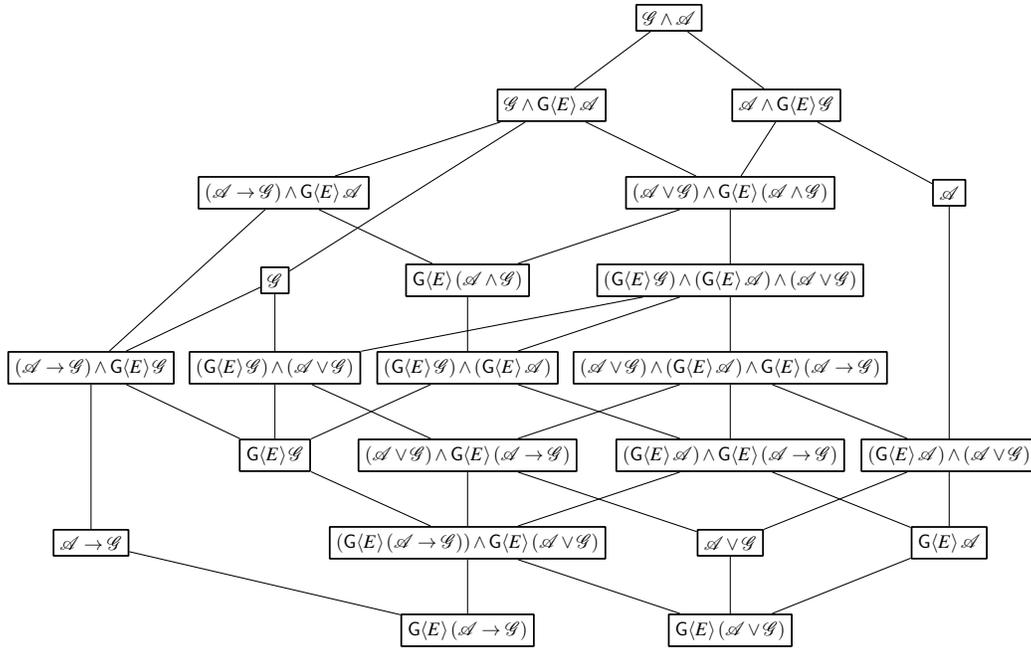- $\mathsf{G}\langle E\rangle \mathscr{G}$ implies $\mathsf{G}\langle E\rangle(\mathscr{A} \vee \mathscr{G})$,

Figure 3: A refined hierarchy of cooperation levels.

- $\mathsf{G}\langle E\rangle\mathscr{A}$ implies $\mathsf{G}\langle E\rangle(\mathscr{A}\vee\mathscr{G})$,

- $\mathscr{A}\to\mathscr{G}$ and $\mathscr{A}\vee\mathscr{G}$ together imply $\mathscr{G}$, and

- $\mathscr{A}\to\mathscr{G}$ and $\mathsf{G}\langle E\rangle(\mathscr{A}\vee\mathscr{G})$ together imply $\mathsf{G}\langle E\rangle\mathscr{G}$.

When further extending the conjuncts from Eqn. 2 with $\langle E\rangle(\mathscr{G})$, $\langle E\rangle(\mathscr{A})$, $\langle E\rangle(\mathscr{A}\wedge\mathscr{G})$, $\langle E\rangle(\mathscr{A}\to\mathscr{G})$ and $\langle E\rangle(\mathscr{A}\vee\mathscr{G})$ we obtain a hierarchy of 77 cooperation levels using the following additional reduction rules:

- $\mathsf{G}\langle E\rangle\varphi$ implies $\langle E\rangle\varphi$ for all $\varphi\in\{\mathscr{G},\mathscr{A},\mathscr{A}\wedge\mathscr{G},\mathscr{A}\to\mathscr{G},\mathscr{A}\vee\mathscr{G}\}$

- $\langle E\rangle\mathscr{G}$ implies $\langle E\rangle(\mathscr{A}\to\mathscr{G})$,

- $\langle E\rangle(\mathscr{A}\wedge\mathscr{G})$ implies $\langle E\rangle\mathscr{A}$ and $\mathsf{G}\langle E\rangle\mathscr{G}$,

- $\mathscr{A}\to\mathscr{G}$ and $\langle E\rangle\mathscr{A}$ together imply $\langle E\rangle(\mathscr{A}\wedge\mathscr{G})$,

- $\mathscr{A}$ and $\langle E\rangle\mathscr{G}$ together imply $\langle E\rangle(\mathscr{A}\wedge\mathscr{G})$,

- $\langle E\rangle(\mathscr{A}\to\mathscr{G})$ and $\mathscr{A}$ together imply $\langle E\rangle(\mathscr{G})$.

- $\langle E\rangle\mathscr{G}$ implies $\langle E\rangle(\mathscr{A}\vee\mathscr{G})$,

- $\langle E\rangle\mathscr{A}$ implies $\langle E\rangle(\mathscr{A}\vee\mathscr{G})$,

- $\mathscr{A}\to\mathscr{G}$ and $\langle E\rangle(\mathscr{A}\vee\mathscr{G})$ together imply $\langle E\rangle\mathscr{G}$.

The first six reduction rules directly correspond to the rules from Section 4.1 but using $\langle E\rangle$ instead of $\mathsf{G}\langle E\rangle$. The latter three rules correspond to rules from the previous paragraph, again instantiated with $\mathsf{G}\langle E\rangle$ instead of $\langle E\rangle$.