How to Handle Assumptions in Synthesis*

Roderick Bloem[1] Rüdiger Ehlers[2,3] Swen Jacobs[1] Robert Könighofer[1]

[1]Graz University of Technology
Graz, Austria

[2]University of Bremen
Bremen, Germany

[3]DFKI GmbH
Bremen, Germany

The increased interest in reactive synthesis over the last decade has led to many improved solutions but also to many new questions. In this paper, we discuss the question of how to deal with assumptions on environment behavior. We present four goals that we think should be met and review several different possibilities that have been proposed. We argue that each of them falls short in at least one aspect.

1 Introduction

In reactive synthesis, we aim to automatically build a system that fulfills guarantees Gua under the assumption that the environment fulfills some properties Ass. Most popular synthesis approaches take the rudimentary view that the system and its environment are adversaries, and that the synthesis problem is solved by generating a system that realizes the formula

$$\mathsf{Ass} \rightarrow \mathsf{Gua}.$$

We argue that this view is imperfect, describe principles that we believe are important to obtain desirable systems, review the work of others who have come to similar conclusions, and describe drawbacks of the proposed approaches. The purpose of the paper is to raise questions rather than to present answers, and to highlight (our) lack of understanding of the problem, rather than our understanding of a solution. Doing this, we hope to spark discussions and further research on this topic.

To see that the setting described above is imperfect, consider a hypothetical example from real life. Suppose that a coach promises the owners of his/her team to win a match under the reasonable assumption that none of the coach's players gets injured during the match. In order to fulfill this contract, the coach may either work hard at winning the game, or may injure one of the players during the last few minutes of the match. While the latter approach may not be unheard of[1], it is generally frowned upon. The same problem occurs in synthesis: a system may fulfill the specification $\mathsf{Ass} \rightarrow \mathsf{Gua}$ by forcing the environment to violate the assumptions, which is quite undesirable [27].

We will assume that systems are implemented in a setting that consists of multiple components. Some of these may be implemented by a synthesis tool and are thus correct by construction. Some may be implemented by a human and we should have good faith in correctness, but not certainty. Some components involve physical interaction with the environment, and we should be skeptical of the assumptions that we have made about these components [32]. The same applies for components whose functionality is carried out by a human operator. Finally, note that even in a perfectly implemented system, errors occur due to environmental influences such as *soft errors* [29].

*This work was supported in part by the Austrian Science Fund (FWF) through the national research network RiSE (S11406-N23) and the project QUAINT (I774-N23), as well as by the European Commission through project STANCE (317753).

[1]Fiorentina's coach (in 2012) and Nuova Cosenza's coach (in 2013) mostly likely each had a different motivation for attacking their own players.

K. Chatterjee, R. Ehlers, and S. Jha (Eds.):
Third Workshop on Synthesis (SYNT 2014)
EPTCS 157, 2014, pp. 34–50, doi:10.4204/EPTCS.157.7

To be sure, the requirement that the system fulfills the guarantees in all cases that the assumptions are fulfilled is very natural and captures the notion of correctness. Yet, it is a very incomplete notion of what is desirable in a system. We present the following (non-exhaustive) list of functional goals that we believe a desirable system should aim for:

Be Correct! Fulfill the guarantees if the environment fulfills the assumptions.

Don't be Lazy! Fulfill the guarantees as well as possible for as many situations as possible, even when the assumptions are not fulfilled.

Never Give Up! If you cannot satisfy the guarantees for every environment behavior, try to satisfy them when you can.

Cooperate! If possible, allow or help the environment to fulfill the assumptions.

Note the difference between *Don't be Lazy* and *Never Give Up*: in the first case, we can enforce the guarantees. In the latter we cannot enforce it, but we may be able to succeed if the environment does not exhibit worst-case behavior. Besides these four functional goals, we want the assumptions to be an abstraction of the environment's specifications so as not to make the synthesis procedure unduly complex.

Traditional LTL synthesis only meets the goal to *Be Correct*. In Section 2, we will show this in more detail, along with the limitations of the approach. After that, we survey and illustrate some existing approaches for the other goals. For *Don't be Lazy* (Sect. 3) we focus on robust and error-resilient synthesis, and on synthesis with quantitative objectives since these approaches attempt to satisfy guarantees as well as possible. For *Never Give Up* (Sect. 4) we will look at research that goes beyond a purely adversarial view of games by suggesting reasonable strategies for losing states. For *Cooperate* (Sect. 5) we will leave the adversarial view even further: We will consider non zero-sum approaches which allow for explicit collaboration by constructing joint strategies for the players. For each approach we will show how it addresses at least one goal and how it is imperfect for another. Finally, in Section 6, we will conclude our investigation with a table summarizing the strengths and weaknesses of the discussed approaches.

2 Be Correct!

2.1 Standard Synthesis

In standard synthesis, the environment is treated as an adversary, i.e., synthesized systems must be correct for *any* possible behavior of the environment. The behaviors of the environment under which the guarantees Gua must hold are then modeled as antecedents to the implication

$$\mathsf{Ass} \rightarrow \mathsf{Gua}.$$

The corresponding payoff matrix[2] is shown in Figure 1: the only one case that is considered undesirable is when Ass is fulfilled, but Gua is not; there is no difference in payoff or desirability between the three remaining cases.

First, the implication does not enforce *Don't Be Lazy*: It does not distinguish a trace that satisfies Gua from one that violates Ass. Thus, it does not restrict the behavior of the system in any way on traces where the environment violates Ass. (An example can be found in Section 3.)

Second, the formalization does not imply the satisfaction of *Never Give Up*. Standard synthesis only optimizes the worst-case behavior, i.e., if $\mathsf{Ass} \rightarrow \mathsf{Gua}$ cannot be fulfilled for some behavior of

[2] In all matrices, the absolute values of the payoffs are immaterial and only illustrate relative preferences.

	Ass violated	Ass satisfied
Gua violated	**1**	**0**
Gua satisfied	**1**	**1**

Figure 1: The standard desirability matrix for satisfying the specification.

the environment, then the output of the synthesis algorithm will simply be "unrealizable", instead of a system that fulfills Gua whenever possible. This goal is all the more important in a situation in which the assumptions are violated. In that case, the guarantees may not be realizable, but even then they should be fulfilled whenever possible. (An example can be found in Section 4.)

Third, the approach does not fulfill *Cooperate*: the system may *force* the environment to violate Ass.

Example 1. Consider the specification

$$(\mathsf{G}\mathsf{F}\, r \wedge \mathsf{G}(r \rightarrow \mathsf{X}(\neg r\, \mathsf{W}\, g))) \;\rightarrow\; \mathsf{G}(r \rightarrow \mathsf{X} g).$$

This specification should result in a system that grants every request in the next time step, for every environment that gives infinitely many requests, but no request is repeated before there is a grant. In this setting, requests are signaled by setting r to true, whereas grants are signaled by setting g to true. By simply giving no grants at all (violating the guarantees), the system can force the environment to violate the assumptions, thus fulfilling the specification. ⋆

The behavior shown in the example may be intended if the environment is considered to be purely adversarial, but in many applications of system design, this is not the case. Good examples for this fact are large systems that are constructed modularly, where the overall system is abstracted by environment assumptions for a particular component that we want to synthesize. Then our goal is not for the component to force the environment (i.e., the rest of the system) to violate the assumptions, but to work together with the environment to some extent, allowing both Ass and Gua to be satisfied whenever possible.

Thus, the standard approach fulfills *Be Correct*, but not *Don't Be Lazy*, *Never Give Up*, and *Cooperate*. This is summarized in Table 5.2 (on page 48) together with the strengths and weaknesses of the approaches discussed in the next sections.

3 Don't Be Lazy!

Traditionally, correctness is considered to be a Boolean property: a system either realizes a specification or not. For specifications of the form Ass → Gua, this attitude results in the desirability matrix shown in Figure 1. This section focuses on improving the system behavior if assumptions are violated, i.e., on the left column of the matrix.

Example 2. As motivation, consider a flight control system which must work correctly under the assumption that the number of simultaneously arriving planes is less than 100. For more planes, the specification may be unrealizable, e.g., because it may be impossible to guarantee all timing constraints. Suppose further that the system has been synthesized, is in operation, and the 101$^{\text{st}}$ plane arrives. A work-to-rule synthesis algorithm could have considered this situation as *won*, and may have randomly

	Ass violated	Ass satisfied
Gua violated	**1**	**0**
Gua satisfied	**2**	**2**

Figure 2: The desirability matrix used in error-resilient synthesis.

chosen to stop serving *any* plane in this situation. A more desirable system would serve planes as well as possible, even though the assumption is violated: For instance, ignoring the 101^{st} plane or responding a bit slower are certainly better options. Even more, for configurations of the 101 planes that can be handled with the available resources, it would be preferable if no reduction in the quality of service occurs at all. ⋆

With the matrix in Figure 1, once the assumptions are violated, there is no additional benefit for the system to satisfy the guarantees any more. The implication is satisfied for any future system behavior, so it can then behave arbitrarily. The synthesis algorithm can exploit this freedom even in situations in which it would still be possible to satisfy the guarantees. This is clearly undesirable. Intuitively, the synthesized system should always aim for satisfying the guarantees, even if assumptions are violated, instead of getting lazy and doing only the least to satisfy the implication.

With the payoff matrix in Figure 2, this changes. By giving traces of the system in which the system satisfies the guarantees a higher payoff regardless of whether the assumptions are satisfied, there is always an incentive for the system to satisfy the guarantees. An approach to deal with multiple ranked specifications is presented in [1].

In practice, assumptions and guarantees can be violated only slightly, or very badly. With this non-Boolean understanding of property violations, the desirability matrix of Figure 2 gets blurred, with gradual transitions between the quadrants, as represented in Figure 3. It makes sense to consider the degree in which guarantees are satisfied also in synthesis: even if it is not possible to satisfy all guarantees due to assumption violations, an ideal system would still try to satisfy guarantees "as well as possible".

In the remainder of this section, we briefly review previous approaches to synthesize systems that are eager to satisfy their guarantees. We start by reviewing the *strict implication semantics* employed in the *Generalized Reactivity(1) Synthesis* approach in Section 3.1, which yields a form of such eagerness as a by-product. In Section 3.2, we then discuss approaches that extend the set of environment behaviors under which the system can satisfy its guarantees, and in this way make the system less lazy without sacrificing the satisfaction of the guarantees. Synthesis approaches that allow slight deviations from the guarantees in case of assumption violations are discussed in Section 3.3. Finally, we discuss quantitative synthesis in Section 3.4, which offers a flexible framework to encode quality criteria of synthesized systems, including some notions of eagerness of the system to satisfy its guarantees.

3.1 Assumptions in Generalized Reactivity Games

Specifications in the generalized reactivity fragment of rank 1 (GR(1)) have been proposed as an alternative to full LTL, as their synthesis problems are efficiently decidable and are still sufficiently expressive for many important properties [8]. What is particularly interesting in our present comparison is that in GR(1) synthesis games that solve the synthesis problem for this fragment, the implication

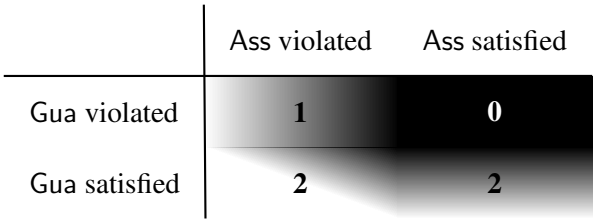

Figure 3: The (blurred) desirability matrix used in robust synthesis.

Ass $\rightarrow$ Gua is interpreted slightly different than in the standard semantics (see Bloem et al. [8] and Klein and Pnueli [27]). In particular, safety guarantees and assumptions are treated differently: even if the environment does not satisfy Ass completely, the system must satisfy its safety guarantees at least as long as the environment satisfies the safety assumptions. This rules out some non-intuitive behavior by the system, where it violates Gua because it knows that it can force the environment to violate Ass at some point in the future. In particular, the unintended behavior in Example 1 is ruled out.

While this changes the *rules* of the synthesis game such that the system player loses the game if such a safety guarantee is violated before the environment violates some safety assumption (instead of the system winning whenever the environment violates Ass anywhere in the infinite trace), it does not change the purely adversarial view on the game.

Example 3. If the safety guarantee $\mathsf{G}(r \rightarrow \mathsf{X}g)$ in Example 1 is changed into a liveness guarantee $\mathsf{G}(r \rightarrow \mathsf{F}g)$, i.e., the specification is modified to

$$(\mathsf{GF}r \wedge \mathsf{G}(r \rightarrow \mathsf{X}(\neg r \,\mathsf{W}\, g))) \;\rightarrow\; \mathsf{G}(r \rightarrow \mathsf{F}g),$$

the system can still enforce an assumption violation by violating the guarantee, even in the modified semantics of the implication, by not giving any grants. The reason is that the system does not violate the guarantee *before* the assumption is violated. $\star$

Furthermore, this extension does not change the purely worst-case analysis that will simply return "unrealizable" if the specification cannot be fulfilled in all cases, and otherwise return a solution that does not distinguish between cases where $\neg$Ass $\wedge \neg$Gua holds versus cases where Gua is actually satisfied. (Recall that strengths and weaknesses are summarized in Table 5.2.)

Related mechanisms are presented in [18]. This work presents an approach to synthesize event-based behavior models from GR(1) specifications. It uses the following definitions in order to avoid systems that satisfy the specification by violating assumptions. A *best effort system* system satisfies the following condition: if the system forces Ass not to hold after a finite trace σ, then no other system that achieves Gua could have allowed Ass after σ. An even stronger definition is that of an *assumption preserving* system: the system should never prevent the environment from fulfilling its assumptions. Every assumption preserving system is also a best effort system. Finally, the authors propose *assumption compatibility* as a methodological guideline. It is a sufficient condition under which any synthesized system is assumption preserving: The environment must be capable of achieving Ass regardless of system behavior. This can be checked by deciding realizability with swapped roles. However, this condition is rather strong.

3.2 Synthesizing Error-Resilient Systems

The most desirable form of the system to react to environment assumption violations is to continue to satisfy its guarantees. As in a system engineering process, assumptions are typically only added on an as-needed basis, this will only be possible in rare circumstances, and the synthesized system can then simply be made robust against assumption violations by removing them before performing synthesis.

Yet, this does not mean that every single assumption violation requires the system to violate its guarantees. A couple of approaches aim at exploiting this fact.

Topcu et al. [32] describe an approach to weaken the safety part of the assumptions as much as possible in context of GR(1) synthesis. The weakening is performed in a very fine-grained way, much finer than how a human specifier would do so, and as fine-grained as possible in GR(1) synthesis without the introduction of additional output signals to encode more complex properties. The resulting synthesized controller is then completely error-resilient against environment behavior that is forbidden by the original assumptions, but allowed by the refined assumptions.

Ehlers and Topcu [20] approach the problem from a different angle. They describe how to synthesize a k-resilient implementation. The notion of k-resilience has been defined earlier by Huang et al. [26]. Adapted to the case of GR(1) specifications, it requires the system to satisfy the guarantees if not more than k safety assumption violations occur in between assumption-violation-free periods of the system execution, provided that these periods are long enough to allow the system to recover. The approach also allows a more fine-grained analysis of for which assumptions some of their violations can be tolerated and for which no violation can be tolerated – whenever there is a trade-off between the choices of assumptions for which violations should be tolerated, all Pareto-optimal such choices are presented to the specifier.

Orthogonal to k-resilient synthesis is the idea to extend a synthesized implementation by *recovery transitions* [34]. Such transitions can be added for cases in which the assumptions are violated, but for which the system can react in a way that does not jeopardize the system's ability to completely satisfy its guarantees along its run if the environment starts to satisfy its assumptions again. In contrast to k-resilient synthesis, recovery behavior is added on a best-effort basis and the synthesized system does not strategically choose its nominal-case behavior such that as many safety assumption violations as possible are tolerated.

All three approaches only make the system robust to a certain extend as they extend the set of environment behaviors under which a system can be synthesized. They do not help to satisfy the guarantees as well as possible for environment behaviors that do not fall into this set.

3.3 Synthesis of Robust Systems

The basic idea of robust synthesis is to satisfy guarantees as well as possible, even if assumptions are violated. Slight violations of the guarantees are allowed when the assumptions are violated, and we can further distinguish between different severity levels of assumption- and guarantee violations.

Robust synthesis is motivated by the observation that synthesized systems sometimes simply stop responding in any useful way after an assumption has been violated. Consider the following example.

Example 4. A system must grant two requests, but not simultaneously: $\mathsf{Gua} = \mathsf{G}((r_0 \rightarrow g_0) \wedge (r_1 \rightarrow g_1) \wedge (\neg g_0 \vee \neg g_1))$. The environment must not raise both requests simultaneously: $\mathsf{Ass} = \mathsf{G}(\neg r_0 \vee \neg r_1)$. The plain implication $\mathsf{Ass} \rightarrow \mathsf{Gua}$ allows the system to ignore any future request if the environment ever happens to raise both requests. Optimizations for other properties like circuit size of the synthesized solution may exploit this freedom. However, a system that ignores one of the simultaneous requests and then continues normally instead of getting lazy would be more preferable. $\star$

Of course, in case of violated assumptions, it may not always be possible to satisfy all guarantees, as Example 4 shows. Otherwise, some assumptions would be superfluous. Also, it makes sense to take the severity of the assumption violation into account. Intuitively, a small assumption violation should also lead to only small guarantee violations. Therefore, the crux in robust synthesis is to define measures of how well guarantees are satisfied and how severe assumptions are violated. Then, an optimal ratio with respect to these metrics can be enforced. Existing approaches [5] typically optimize the worst case of this ratio. For safety properties, a natural conformance measure for both assumptions and guarantees is to count the number of time steps in which properties are violated. For liveness properties, this does not work because a liveness property violation cannot be detected at any point in time: If some event is supposed to happen eventually, and has not happened yet, we may just not have waited long enough. If Ass and Gua are composed of several properties, one can also count the number of violated properties to define the severity of a violation [5].

Despite the fact that liveness assumption violations cannot be observed at runtime, robust synthesis approaches for specifications with liveness assumptions and guarantees exist that let the system tolerate (safety) assumption violations. Intuitively, the idea is to ask the system to tolerate safety assumption violations if in only finitely many steps of the system's execution, such violations occur. The system is then only allowed to violate safety guarantees finitely often. Liveness assumptions are assumed to hold at all times. Since the system cannot know when an assumption violation has been the last one, it has to behave in a robust way [19]. As a variant to the approach, the system can additionally be required to satisfy the liveness guarantees even if safety assumptions are violated infinitely often [7].

In summary, robustness is definitely a useful extension to correctness. One shortcoming of existing solutions is that they only optimize the robustness measure for the worst case, i.e., assume a perfectly antagonistic environment. As a consequence, the resulting system may still be unnecessarily lazy for more cooperative environment behaviors. The fact that the system cannot satisfy the guarantees any better in the worst case should not be an excuse for not trying. In this sense, not assuming a fully adversarial environment in the robustness optimization may yield even better results. This aspect will be elaborated in Section 4.

3.4 Quantitative Synthesis

Among all systems that realize a given specification, some may be more desirable than others. The idea of synthesis with quantitative objectives is to construct a system that not only satisfies the (qualitative) specification, but also maximizes a (quantitative) desirability metric. In this sense, some approaches to robust synthesis, as discussed in the previous section, can be seen as special cases of quantitative synthesis. But quantitative synthesis can also be a handy tool to optimize solutions with respect to other desirability metrics.

Example 5. Continuing Example 4, we may prefer systems that give as few unnecessary grants as possible. This can be achieved by assigning costs to unnecessary grants (i.e., situations with $g_i \wedge \neg r_i$), and let the synthesis algorithm minimize these costs. ⋆

Of course, one could also specify each and every situation where no grant should be given. While this is quite possible for this small example, it can be tedious, error-prone, and destroy the abstract quality of the specification for more complicated cases: Ideally, a specification only expresses *what* the system should do, but not *how*. If the exact behavior needs to be specified for each and every situation, it is better to implement the system right away.

The work of [6] presents a machinery based on games with a lexicographic mean-payoff objective and a simultaneously considered parity objective to solve such problems. The parity objective encodes

the qualitative specification, while the mean-payoff objective encodes the quantitative desirability metric. The approach assumes fully adversarial environments and optimizes for this worst case.

Defining a desirability metric for a system is never an easy task. Cerný and Henzinger [11] propose to define it in two steps. The first step is to assign costs (or payoffs) to single traces. This can be done by combining the costs of single events in the trace, e.g., by taking the sum, average, maximum, etc. Second, the costs for individual traces are combined into total costs. Again, there are various options like taking the worst case, the average case, or a weighted average assuming some probability distribution. Although this approach is quite generic, it is questionable if the desirability of a system can be expressed by one single number in the venture of satisfying guarantees as good as possible in as many situations as possible. Dominance relations inducing a partial order between systems, as used in the next section, may be a more natural notion as they provide a natural quantification over environment behaviors.

If cost notions for both the environment and system actions can be given, there is a canonical way to define which system traces are desirable: the ones that are the cheapest. Tabuada et al. [31] adapt notions from control theory to define a preferability relation on system behaviors. In addition to minimizing the ratio between environment behavior cost and system behavior cost, they also require that the effect of sporadic disturbances vanishes over time.

Finally, there are approaches that combine quantitative approaches with a *probabilistic* model of the environment, to find the best solution under a given probability distribution for actions of the environment [21]. A combination of probabilistic and worst-case reasoning is considered by Bruyère et al. [10].

In summary, quantitative synthesis does not directly address the problem of dealing with assumptions in synthesis, but can rather be seen as a tool for obtaining better solutions with respect to different metrics. The fact that the environment is considered as perfectly adversarial in most methods may not be ideal in all settings.

4 Never Give Up!

Traditional games-based synthesis is only concerned with the worst case. As already raised in the previous sections, this mind-set is not always justified.

Example 6. The flight control system from Example 2 may actually be able to handle way more than 100 planes in time if they do not all signal an emergency at the same time. This worst case is possible, but very unlikely to happen in practice. ⋆

If a guarantee cannot be enforced in the worst case, traditional synthesis methods will consider this guarantee as "impossible" to achieve. Thus, the constructed system would behave arbitrary if it ever gets into such a "hopeless" situation, i.e., it would not even try to reach the goal. However, when the system is in operation, its concrete environment may not be perfectly adversarial, i.e., the worst case may not occur. Hence, it makes sense for the system to behave faithfully even in (worst-case-)lost situations instead of resigning. In other words, the synthesized system should retain or even maximize the chances of reaching the goal (e.g., satisfying all guarantees even if assumptions are violated), even if this is not possible in the worst case.

Note the difference to robust and quantitative synthesis, as discussed in the previous section: Robust and quantitative synthesis aim at satisfying guarantees as well as possible for the worst case environment behavior. In contrast, this section is concerned with satisfying the specification (preferably without cutbacks) for many environment behaviors that do not represent the worst case as they violate Ass. In the following, we will discuss existing synthesis approaches that tackle this problem by dealing with "hopeless" situations in a constructive way.

4.1 Environment Assumptions

When we consider the basic idea of "restricting" the environment behavior by adding assumptions to an LTL specification of the form $\mathsf{Ass} \rightarrow \mathsf{Gua}$, then synthesis from such a specification results in a system that is guaranteed to satisfy Gua for all behaviors of the environment that satisfy Ass. On the other hand, the system does not give any guarantees for traces on which Ass is violated.

Chatterjee, Henzinger and Jobstmann [14] show how, for a given unrealizable system specification Gua, one can compute an environment assumption Ass, such that $\mathsf{Ass} \rightarrow \mathsf{Gua}$ is realizable (for ω-regular specifications). The computed assumptions consist of a safety and a liveness part, and should be as weak as possible. While *minimal* (but not unique) safety assumptions[3] can be computed efficiently, the problem is NP-hard for minimal liveness assumptions[4]. If it is sufficient to compute a *locally minimal* set of liveness assumptions, i.e., a set of liveness assumptions from which no element may be removed without changing the resulting specification to be realizable, NP-hardness can be avoided.

4.2 Best-Effort Strategies for Losing States

Faella [23, 22] investigates best-effort strategies for states from which the winning condition cannot be enforced. Intuitively, a good strategy should behave rationally in the sense that it does not "give up". Hence, this work assumes the desirability matrix of Figure 1, and is concerned with staying away from the top-right corner, even if this is not possible in the worst case.

Example 7. As an example, consider the specification $\mathsf{GF}(o \wedge \mathsf{X}\, i)$, where i is an input and o is an output. There is no way the system can enforce satisfying the property. However, setting o to true as often as possible is more promising than setting o always to false. ⋆

Faella [23] discusses and compares several goal independent criteria for such rational strategies. The work concludes that *admissible* strategies, defined via a dominance relation, may be a good choice. Intuitively, strategy σ *dominates* strategy σ' if σ is always at least as good as σ', and better for at least one case. More specifically, σ dominates σ' if (1) for all environment strategies and starting states, if σ' satisfies the specification then σ does so too, and (2) there exists some environment strategy and starting state from which σ satisfies the specification but σ' does not. This induces a partial order between strategies. An *admissible* strategy is one that is not dominated by any other strategy.

For positional[5] and prefix-independent[6] goals, Faella [23] presents an efficient way to compute admissible strategies: the conventional winning strategy σ is computed and played from all winning states. For the remaining states, a cooperatively winning strategy σ' is computed, assuming that σ is played in the winning states. This is a very relevant result because, e.g., parity goals are positional and prefix-independent, and LTL specifications can be transformed into parity games. For goals that do not fall into this category, the computation of admissible strategies is left for future work. Unfortunately, this work has not been actively followed up on.

Damm and Finkbeiner also consider admissible strategies, called *dominant strategies* in [17], and show that for a non-distributed system, a dominant strategy can be found (or its non-existence proved) in 2EXPTIME. That is, dominant strategies are not harder to find than the usual winning strategies. Since

[3] *Minimal* here means that a minimal number of environment edges are removed from the game graph.

[4] Here, *minimal* means to put fairness conditions on a minimum number of environment edges in the game graph.

[5] A goal is positional if the strategy does not require memory on top of knowing the current position in *synthesis games* that are built from the given specification.

[6] A goal is prefix-independent if adding or removing a finite prefix to/from the execution does not render a satisfied property violated.

a dominant strategy must be winning if a winning strategy exists, this means we can find best-effort strategies in the same time complexity as usual winning strategies, without sacrificing the basic goal of correctness.

The focus of [17] is however on the synthesis of dominant strategies for systems with multiple processes, which is shown to be effectively decidable (with a much lower complexity than with other approaches) for specifications that are known to have dominant strategies. Moreover, the constructed strategies are modular, and synthesis can even be made compositional for safety properties. Thus, in this case we not only obtain strategies that do their best even if the specification cannot always be fulfilled, but we can find such a strategy even in cases where the classical distributed synthesis problem is undecidable.

Even though it is in some sense orthogonal to our question of how to properly treat assumptions, we view the behavior of the system on lost states as an important ingredient to building desirable systems. In a system composed of components that are not necessarily adversarial, this approach may help reach a common goal. While robust synthesis attempts to satisfy guarantees as well as possible under the worst-case environment, the best-effort strategies attempt to increase the chances of satisfying all guarantees under a friendly environment assumption. Both views have their merits.

4.3 Fallback to Human

Another interesting way of dealing with "hopeless" situations in synthesis has recently been presented by Li et al. [28]. Safety critical control systems like autopilots in a plane or driving assistance in a car are usually not fully autonomous but involve human operators. If the environment behaves such that guarantees cannot be enforced, the controller can therefore simply ask the human operator for intervention. This allows for semi-autonomous controllers, even for unrealizable specifications. There are two additional requirements: the human operator should be notified ahead of time, and no unnecessary intervention should be required.

The approach computes a non-deterministic counterstrategy. In operation, the controller constantly monitors the behavior of the environment and tracks if it conforms to this counterstrategy. This prevents alarms when the environment is not fully adversarial, so that the guarantees can be enforced even though the specification is unrealizable in the worst case. Notifying the human operator ahead of a potential specification violation is achieved by requiring a minimum distance (number of steps) to any failure-prone state.

The faithfulness of this approach is similar in spirit to the best-effort strategies discussed in the previous section: the specification cannot be satisfied in the worst case, but this should not be an excuse for resigning. The worst case may not occur (often) in operation, and the synthesized system should take advantage of this. While requiring human intervention may only be an option in specific settings, the idea of checking the actual environment behavior against a counterstrategy in order to assess whether the environment is behaving in an adversarial manner is definitely interesting.

4.4 Markov Decision Processes

Another way of refraining from worst case assumptions in synthesis is by using Markov Decision Processes (MDPs) [4, 2]. The environment is not considered to behave adversarially but randomly with a certain probability distribution. This situation is also referred to as *1.5 player game* (the probabilistic environment only counts as half a player). Strategies for such games attempt to maximize the probability to satisfy the goal. There also exist solutions to maximize quantitative objectives against a random

	Ass violated	Ass satisfied
Gua violated	**0**	**0.25**
Gua satisfied	**0.75**	**1**

Figure 4: Cooperative desirability matrix.

player [15].

MDPs as the sole synthesis algorithm may not be satisfactory since optimality against a random player does not necessarily imply that the strategy is winning against an adversarial player [23]. Nevertheless, MDPs can be valuable to optimize the behavior in lost states, or to specialize a winning strategy that allows for multiple options in several situations.

5 Cooperate!

Realistic applications of synthesis methods will in general not synthesize a complete system from scratch, but will separate the system into components that can be implemented (either by hand or by synthesis) modularly. To make such an approach tractable while still giving global correctness guarantees, synthesis of every component must take into account the expected behavior of the rest of the system, again expressed as some kind of environment assumption.

Thus far, we have discussed synthesis approaches that are designed to prefer cases where Gua is satisfied over cases where Ass is violated (Sect. 3), and that try to optimize the result even if the goal cannot be reached in all cases (Sect. 4). In some sense, the latter can be seen as an implicit collaboration with the environment, i.e., *hoping* that it is not its main goal to hurt us.

In this section, we consider synthesis algorithms for systems that *explicitly* cooperate. In this case, the environment can really be considered as a second system player, and the payoff matrix is notably different, see Figure 4. In particular, we do not want "our" system component to force assumption violations in other system components, as this would lead to incorrect behavior of the overall system. Instead, we want synthesis to be based on a "good neighbor assumption", i.e., the environment will only violate the assumptions if necessary, and we should not force it to do so, but try to make the overall system work even if the assumptions are not always satisfied.

The basic idea is that environment and system can cooperate to some extent, in order to satisfy both Ass and Gua. If we allow *full cooperation*, then the synthesis problem becomes the problem of synthesizing an implementation for both the environment and the system, and requiring them to jointly satisfy $\mathsf{Ass} \wedge \mathsf{Gua}$. This problem has been considered for different models of communication [16, 30, 24]. Such solutions are however unsatisfactory for two reasons:

(i) The approaches synthesize one particular implementation of the environment. This will only be a correct implementation in the overall system if Ass contains *all* of the required properties of the rest of the system, not allowing us to abstract from parts of the environment.

(ii) The synthesized implementation of the system is guaranteed to satisfy Gua only for *exactly this environment*. Thus, the approach does also not allow additional refinement or modification of the environment behavior.

Together, these two properties imply that we cannot use such an approach to modularize synthesis, as we need to synthesize both components in full detail at the same time.

In the following, we consider *assume-guarantee synthesis* (Sect. 5.1) and *synthesis under rationality assumptions* (Sect. 5.2), two approaches that are between a completely adversarial and a completely cooperative environment behavior. Both are based on the notion of non-zero-sum games, i.e., games in which players do not have mutually exclusive objectives, but can reach (part of) their respective objectives by cooperation.

5.1 Assume-Guarantee Synthesis

Intuitively, the *assume-guarantee synthesis* approach by Chatterjee and Henzinger [13] wants to synthesize implementations for two parallel processes P_1, P_2 (which could be the system and the environment) such that solutions are robust with respect to changes in the other process, as long as it does not violate its own specification. More formally, we want to find implementations of P_1, P_2 that satisfy $\phi_1 \wedge \phi_2$ together, and furthermore the solutions should be such that each process P_i satisfies $\phi_j \rightarrow \phi_i$ for *any* implementation of the other process P_j. That is, given a pair of solutions for P_1, P_2, we can replace one of them with a different implementation. As long as it satisfies its own specification ϕ_j (together with the fixed implementation for the other process), we know that the overall specification $\phi_1 \wedge \phi_2$ will still hold.

This means that players have to cooperate to find a common solution, but cooperation is also limited, in that the players cannot decide on one particular strategy to satisfy the joint specification. Thus, assume-guarantee synthesis is an option *between* purely adversarial and purely cooperational synthesis: if we obtain process implementations P_1 and P_2 that satisfy $P_i \models \phi_j \rightarrow \phi_i$ for adversarial synthesis, then the parallel composition $P_1 \parallel P_2$ of these two implementations will also satisfy the conjunction $\phi_1 \wedge \phi_2$. Since each of them satisfy their spec in an arbitrary environment, they in particular satisfy the assume-guarantee specification. Moreover, every solution for assume-guarantee synthesis obviously is also a solution for cooperative synthesis.

Example 8. Consider two processes P_1, P_2, each with one output o_i that can be read by the other process, and specifications

$$\phi_1 = \left\{ \begin{array}{ll} & \mathsf{G}\mathsf{F}\, o_1 \\ \wedge & \mathsf{G}(o_1 \rightarrow \mathsf{X}\neg o_1) \end{array} \right\}, \quad \phi_2 = \mathsf{G}((\mathsf{X}\, o_2) \leftrightarrow o_1)\,.$$

There are several implementations for P_1 that satisfy ϕ_1 (and do not depend on the implementation of P_2), and several implementations for P_2 that satisfy ϕ_2, most of them depending on the implementation of P_1. For example, P_1 might raise o_1 in the initial state, and then every third tick. For this implementation, a suitable implementation for P_2 can raise o_2 in the first tick after the initial state, and then every third tick from there.

While this implementation for P_2 is correct for the particular implementation of P_1, it is not correct for all implementations of P_1 that satisfy ϕ_1. For example, P_1 could raise o_1 every second tick, and the given P_2 would not satisfy ϕ_2 anymore. However, there is an implementation that satisfies ϕ_2 for all implementations of P_1 that satisfy ϕ_1: P_2 can simply read o_1 and go to a state where it raises o_2 iff o_1 it currently active. Only such a solution for P_2 solves the assume-guarantee synthesis problem (any solution for P_1 that satisfies ϕ_1 is fine, since it does not depend on P_2).

Furthermore, consider the extended specification

$$\phi_1 = \left\{ \begin{array}{ll} & \mathsf{G}\mathsf{F}\, o_1 \\ \wedge & \mathsf{G}(o_1 \rightarrow \mathsf{X}\neg o_1) \end{array} \right\}, \quad \phi_2 = \left\{ \begin{array}{ll} & \mathsf{G}((\mathsf{X}\, o_2) \leftrightarrow o_1) \\ \wedge & \mathsf{G}(\neg o_2 \rightarrow \mathsf{X}\, o_2) \end{array} \right\}.$$

Now, while there are implementations for both processes with $(P_1 \parallel P_2) \models \phi_1 \wedge \phi_2$, there is no solution of the assume-guarantee synthesis problem: a solution for P_2 *must* raise o_2 at least every second tick now, and will only work with such implementations of P_1, but not with those that raise o_1 less frequently (even if they still satisfy ϕ_1). ⋆

5.2 Synthesis under Rationality Assumptions

A number of different approaches to the synthesis of multi-component systems relies on the notion of *rationality*. Informally, this means that every component has a goal that it wants to achieve, or a payoff it wants to maximize, and it will always use a strategy that maximizes its own payoff. Both the rationality of players and the goals of all components are assumed to be common knowledge. In particular, a player will only use a strategy that hurts other components if this will not lead to a smaller payoff for itself. As can be expected, this leads to implementations that do not behave purely adversarial, but cooperate to some degree in order to satisfy their own specification.

We survey three different approaches based on rationality: *rational synthesis* by Fisman, Kupferman and Lustig [25], methods based on *iterated admissibility* by Berwanger [3] and by Bernguier, Raskin and Sassolas [9], and an extension of the notion of secure equilibria to the multi-player case, called *doomsday equilibria* [12].

Rational Synthesis. The *rational synthesis* approach centers synthesis around a special *system process*, and produces not only an implementation for the system, but also strategies for all components in the environment, such that the specification of the system is satisfied, and the strategies of the components are optimal in some sense. To guarantee correctness, the approach assumes that these strategies can be communicated to the other components, and that the components will not use a different strategy than the one proposed, as long as it is optimal.

The definition of what is considered to be an optimal strategy leaves some freedom to the approach. The authors explore Nash equilibria (cp. Ummels [33]), dominant strategies (cp. Faella [23], Damm and Finkbeiner [17]), and subgame-prefect Nash equilibria (also [33]). Intuitively,

- if the set of proposed strategies is a Nash equilibrium profile, then no process can achieve a better result if it changes its strategy (while all others keep their strategies);
- if the set of strategies is a dominant strategy profile, then no process can achieve a better result if any number of processes (including itself) change their strategy;
- if the set of strategies is a subgame-perfect equilibrium profile, then no process can achieve a better result for any arbitrary history of the game[7] by changing its strategy (while all others keep their strategies).

Compared to assume-guarantee synthesis, this approach does not guarantee that the synthesized implementation will also work when other processes change their behavior, even if the different behavior still satisfies the specification. Instead, it is based on the assumption that other processes have no incentive to change their behavior, which is somewhat unsatisfactory for a modular synthesis approach.

Example 9. Consider again the example from above,

$$\phi_1 = \left\{ \begin{array}{ll} & \mathsf{G}\mathsf{F}\, o_1 \\ \wedge & \mathsf{G}(o_1 \rightarrow \mathsf{X}\neg o_1) \end{array} \right\}, \quad \phi_2 = \left\{ \begin{array}{ll} & \mathsf{G}((\mathsf{X}\, o_2) \leftrightarrow o_1) \\ \wedge & \mathsf{G}(\neg o_2 \rightarrow \mathsf{X}\, o_2) \end{array} \right\}.$$

[7] even those that do not correspond to the given strategy profile

A solution that satisfies $(P_1 \parallel P_2) \models \phi_1 \wedge \phi_2$ is also a rational synthesis solution, for any of the notions of optimality above. However, for a Nash equilibrium, a pair of implementations for P_1, P_2 is also a solution if $(P_1 \parallel P_2) \models \phi_1$ and $(P_1 \parallel P_2) \models \neg\phi_2$, as long as there does not exist an implementation P_2' for which $(P_1 \parallel P_2') \models \phi_2$. $\star$

Rational synthesis with Nash equilibrium has strictly weaker conditions on implementations than assume-guarantee synthesis. That is, any solution of assume-guarantee synthesis will also be a solution for this case of rational synthesis, but this is not always the case in the other direction. Also, dominant or subgame-perfect equilibria strategy profiles will always be Nash equilibrium profiles, but the set of solutions seems to be incomparable with assume-guarantee synthesis.

A combination of assume-guarantee reasoning with rational synthesis seems possible: instead of requiring that the system implementation works exactly in the given equilibrium, it should work for any behavior of the other processes that does not reduce their payoff, or respectively any behavior where they still satisfy their own specification.

Iterated Admissibility. The basic idea of iterated admissibility approaches [3, 9] is similar to rational synthesis: every component has its own goal in a (non-zero-sum) game, and is assumed to be rational in that it avoids strategies that are dominated by other strategies (taking into account all possible strategies of the other players). This avoidance of dominated strategies removes some of the possible behaviors for all players. Both the rationality assumption and the full state of the game being played are assumed to be common knowledge, so every player knows which strategies the other players will eliminate. Under the new sets of possible behaviors, there may be new strategies that are dominated by others, so the process of removing dominated strategies can be iterated and repeated up to a fixpoint.

The basic notions of this class of infinite multi-player games have been defined by Berwanger [3]. Brenguier, Raskin and Sassolas [9] have recently investigated the complexity of iterated admissibility for different classes of objectives, and showed that in general it is similar to the complexity of Nash equilibria.

Compared to rational synthesis, where the system process can compute strategies for all other components and they will accept them if they are optimal, in this case there is no distinguished process. Instead, all processes compute a set of optimal (or admissible) strategies, with full information allowing all components to come to the same conclusions.

Doomsday Equilibria. The notion of *doomsday equilibria* by Chatterjee et al. [12] uses the rationality assumption like the two approaches mentioned before, but takes the punishment for deviating from a winning strategy to the extreme: a doomsday equilibrium is a strategy such that all players satisfy their objective, and if any coalition of players deviates from their strategy and violates the objective of at least one of the other players, then the game is doomed, i.e., the losing player(s) have a strategy such that none of the other players can satisfy their objective.

A distinguishing feature of doomsday equilibria is that their existence is decidable even in partial information settings, in contrast to the other existing notions of equilibria. In the case of two players, doomsday equilibria coincide with the well-known notion of secure equilibria.

6 Conclusions

In this paper, we discussed the role of environment assumptions in synthesis of reactive systems, and how existing approaches handle such assumptions. Besides correctness, we proposed three more properties

Table 1: Comparison of existing approaches.

	Section	Be Correct!	Don't be Lazy!	Never Give Up!	Cooperate!
Ass $\rightarrow$ Gua	2.1	✔			
Strict Realizability	3.1	✔			(✓)
Error-Resilience / Recovery Transitions	3.2	✔	(✓)		
Robustness	3.3	✔	✔		
Quantitative Synthesis	3.4	(✓)	✔		
Synthesizing Environment Assumptions	4.1	✔		✔	
Best Effort Strategies	4.2	✔		✔	
Fallback to Human Control	4.3	(✓)		✔	
Markov Decision Processes	4.4			✔	
Assume-Guarantee Synthesis	5.1	✔			✔
Synthesis under Rationality Assumptions	5.2		(✓)	(✓)	✔

that a good system should realize: systems should satisfy guarantees as well as possible even if environment assumptions are violated (*Don't be Lazy!*), they should aim for satisfying the guarantees even if this is not possible in the worst case (*Never Give Up!*), and systems should rather help the environment satisfy the assumptions instead of trying to enforce their violation (*Cooperate!*). These properties are especially important in modular synthesis, where assumptions are used to abstract other parts of the system rather than expressing "don't care"-situations. As summarized in Table 1, we conclude that none of the existing approaches satisfies all these requirements. Although important steps towards synthesis of high quality systems have been made, we believe that even better results can be achieved by combining and extending ideas from the different branches. *The* perfect solution may not exist, since it may strongly depend on the application. Even if it does exist, it may be prohibitively expensive to achieve. In any case, more research is needed to explore both the most important objectives and the best possible solutions.

References

[1] Rajeev Alur, Aditya Kanade & Gera Weiss (2008): *Ranking Automata and Games for Prioritized Requirements*. In: *Computer Aided Verification (CAV'08), LNCS* 5123, Springer, pp. 240–253, doi:10.1007/978-3-540-70545-1_23.

[2] Christel Baier, Marcus Größer, Martin Leucker, Benedikt Bollig & Frank Ciesinski (2004): *Controller Synthesis for Probabilistic Systems*. In: *Exploring New Frontiers of Theoretical Informatics / Theoretical Computer Science (IFIP/TCS'04)*, Kluwer, pp. 493–506, doi:10.1007/1-4020-8141-3_38.

[3] Dietmar Berwanger (2007): *Admissibility in Infinite Games*. In: *Symposium on Theoretical Aspects of Computer Science (STACS'07)*, *LNCS* 4393, Springer, pp. 188–199, doi:10.1007/978-3-540-70918-3_17.

[4] Andrea Bianco & Luca de Alfaro (1995): *Model Checking of Probabalistic and Nondeterministic Systems*. In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, *LNCS* 1026, Springer, pp. 499–513, doi:10.1007/3-540-60692-0_70.

[5] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer & Robert Könighofer (2014): *Synthesizing robust systems*. *Acta Inf.* 51(3-4), pp. 193–220, doi:10.1007/s00236-013-0191-5.

[6] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2009): *Better Quality in Synthesis through Quantitative Objectives*. In: *Computer Aided Verification (CAV'09)*, *LNCS* 5643, Springer, pp. 140–156, doi:10.1007/978-3-642-02658-4_14.

[7] Roderick Bloem, Hans-Jürgen Gamauf, Georg Hofferek, Bettina Könighofer & Robert Könighofer (2012): *Synthesizing Robust Systems with RATSY*. In: *Workshop on Synthesis (SYNT'12)*, *EPTCS* 84, pp. 47–53, doi:10.4204/EPTCS.84.4.

[8] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Yaniv Sa'ar (2012): *Synthesis of Reactive(1) designs*. *J. Comput. Syst. Sci.* 78(3), pp. 911–938, doi:10.1016/j.jcss.2011.08.007.

[9] Romain Brenguier, Jean-François Raskin & Mathieu Sassolas (2014): *The Complexity of Admissibility in Omega-Regular Games*. In: *Computer Science Logic / Logic in Computer Science (CSL-LICS'14)*, IEEE. To appear.

[10] Véronique Bruyère, Emmanuel Filiot, Mickael Randour & Jean-François Raskin (2014): *Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games*. In: *Symposium on Theoretical Aspects of Computer Science (STACS'14)*, *LIPIcs* 25, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 199–213, doi:10.4230/LIPIcs.STACS.2014.199.

[11] Pavol Cerný & Thomas A. Henzinger (2011): *From boolean to quantitative synthesis*. In: *International Conference on Embedded Software (EMSOFT'11)*, ACM, pp. 149–154, doi:10.1145/2038642.2038666.

[12] Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot & Jean-François Raskin (2014): *Doomsday Equilibria for Omega-Regular Games*. In: *Verification, Model Checking, and Abstract Interpretation (VMCAI'14)*, *LNCS* 8318, Springer, pp. 78–97, doi:10.1007/978-3-642-54013-4_5.

[13] Krishnendu Chatterjee & Thomas A. Henzinger (2007): *Assume-Guarantee Synthesis*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, *LNCS* 4424, Springer, pp. 261–275, doi:10.1007/978-3-540-71209-1_21.

[14] Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2008): *Environment Assumptions for Synthesis*. In: *Concurrency Theory (CONCUR'08)*, *LNCS* 5201, Springer, pp. 147–161, doi:10.1007/978-3-540-85361-9_14.

[15] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann & Rohit Singh (2010): *Measuring and Synthesizing Systems in Probabilistic Environments*. In: *Computer Aided Verification (CAV'10)*, *LNCS* 6174, pp. 380–395, doi:10.1007/978-3-642-14295-6_34.

[16] Edmund M. Clarke & E. Allen Emerson (1981): *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In: *Logic of Programs*, *LNCS* 131, Springer, pp. 52–71, doi:10.1007/BFb0025774.

[17] Werner Damm & Bernd Finkbeiner (2014): *Automatic Compositional Synthesis of Distributed Systems*. In: *Formal Methods (FM'14)*, *LNCS* 8442, Springer, pp. 179–193, doi:10.1007/978-3-319-06410-9_13.

[18] N. D'Ippolito, V. A. Braberman, N. Piterman & S. Uchitel (2013): *Synthesizing nonanomalous event-based controllers for liveness goals*. *ACM Trans. Softw. Eng. Methodol.* 22(1), p. 9, doi:10.1145/2430536.2430543.

[19] Rüdiger Ehlers (2011): *Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis*. In: *NASA Formal Methods*, *LNCS* 6617, Springer, pp. 101–115, doi:10.1007/978-3-642-20398-5_9.

[20] Rüdiger Ehlers & Ufuk Topcu (2014): *Resilience to intermittent assumption violations in reactive synthesis*. In: *Hybrid Systems: Computation and Control (HSCC'14)*, ACM, pp. 203–212, doi:10.1145/2562059.2562128.

[21] Christian von Essen & Barbara Jobstmann (2012): *Synthesizing Efficient Controllers*. In: *Verification, Model Checking, and Abstract Interpretation (VMCAI'12)*, *LNCS* 7148, Springer, pp. 428–444, doi:10.1007/978-3-642-27940-9_28.

[22] Marco Faella (2007): *Games You Cannot Win*. In: *Workshop on Games and Automata for Synthesis and Validation*, Lausanne, Switzerland.

[23] Marco Faella (2009): *Admissible Strategies in Infinite Games over Graphs*. In: *Mathematical Foundations of Computer Science (MFCS'09)*, *LNCS* 5734, Springer, pp. 307–318, doi:10.1007/978-3-642-03816-7_27.

[24] Bernd Finkbeiner & Sven Schewe (2005): *Uniform Distributed Synthesis*. In: *Logic in Computer Science (LICS'05)*, IEEE Computer Society, pp. 321–330, doi:10.1109/LICS.2005.53.

[25] Dana Fisman, Orna Kupferman & Yoad Lustig (2010): *Rational Synthesis*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, *LNCS* 6015, Springer, pp. 190–204, doi:10.1007/978-3-642-12002-2_16.

[26] Chung-Hao Huang, Doron Peled, Sven Schewe & Farn Wang (2012): *Rapid Recovery for Systems with Scarce Faults*. In: *Games, Automata, Logics and Formal Verification (GandALF'12)*, *EPTCS* 96, pp. 15–28, doi:10.4204/EPTCS.96.2.

[27] Uri Klein & Amir Pnueli (2010): *Revisiting Synthesis of GR(1) Specifications*. In: *Haifa Verification Conference (HVC'10)*, *LNCS* 6504, Springer, pp. 161–181, doi:10.1007/978-3-642-19583-9_16.

[28] Wenchao Li, Dorsa Sadigh, S. Shankar Sastry & Sanjit A. Seshia (2014): *Synthesis for Human-in-the-Loop Control Systems*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, *LNCS* 8413, Springer, pp. 470–484, doi:10.1007/978-3-642-54862-8_40.

[29] T.C. May & Murray H. Woods (1979): *Alpha-particle-induced soft errors in dynamic memories*. *Electron Devices, IEEE Transactions on* 26(1), pp. 2–9, doi:10.1109/T-ED.1979.19370.

[30] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *Foundations of Computer Science (FOCS'90)*, IEEE Computer Society, pp. 746–757, doi:10.1109/FSCS.1990.89597.

[31] Paulo Tabuada, Ayca Balkan, Sina Y. Caliskan, Yasser Shoukry & Rupak Majumdar (2012): *Input-output robustness for discrete systems*. In: *International Conference on Embedded Software (EMSOFT'12)*, ACM, pp. 217–226, doi:10.1145/2380356.2380396.

[32] Ufuk Topcu, Necmiye Ozay, Jun Liu & Richard M. Murray (2012): *On synthesizing robust discrete controllers under modeling uncertainty*. In: *Hybrid Systems: Computation and Control (HSCC'12)*, ACM, pp. 85–94, doi:10.1145/2185632.2185648.

[33] Michael Ummels (2006): *Rational Behaviour and Strategy Construction in Infinite Multiplayer Games*. In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, *LNCS* 4337, Springer, pp. 212–223, doi:10.1007/11944836_21.

[34] Kai Weng Wong, Rüdiger Ehlers & Hadas Kress-Gazit (2014): *Correct High-level Robot Behavior in Environments with Unexpected Events*. In: *Robotics: Science and Systems Conference (RSS'14)*, IEEE. To appear.