Reactive Control Improvisation

Daniel J. Fremont(✉) and Sanjit A. Seshia

University of California, Berkeley, USA
{dfremont,sseshia}@berkeley.edu

Abstract. Reactive synthesis is a paradigm for automatically building correct-by-construction systems that interact with an unknown or adversarial environment. We study how to do reactive synthesis when part of the specification of the system is that its behavior should be *random*. Randomness can be useful, for example, in a network protocol fuzz tester whose output should be varied, or a planner for a surveillance robot whose route should be unpredictable. However, existing reactive synthesis techniques do not provide a way to ensure random behavior while maintaining functional correctness. Towards this end, we generalize the recently-proposed framework of *control improvisation* (CI) to add reactivity. The resulting framework of *reactive control improvisation* provides a natural way to integrate a randomness requirement with the usual functional specifications of reactive synthesis over a finite window. We theoretically characterize when such problems are realizable, and give a general method for solving them. For specifications given by reachability or safety games or by deterministic finite automata, our method yields a polynomial-time synthesis algorithm. For various other types of specifications including temporal logic formulas, we obtain a polynomial-space algorithm and prove matching PSPACE-hardness results. We show that all of these randomized variants of reactive synthesis are no harder in a complexity-theoretic sense than their non-randomized counterparts.

1 Introduction

Many interesting programs, including protocol handlers, task planners, and concurrent software generally, are *open* systems that interact over time with an external environment. Synthesis of such *reactive systems* requires finding an implementation that satisfies the desired specification no matter what the environment does. This problem, *reactive synthesis*, has a long history (see [7] for a survey). Reactive synthesis from temporal logic specifications [19] has been particularly well-studied and is being increasingly used in applications such as hardware synthesis [3] and robotic task planning [15].

In this paper, we investigate how to synthesize reactive systems with *random behavior*: in fact, systems where *being random in a prescribed way is part of their specification*. This is in contrast to prior work on stochastic games where randomness is used to model uncertain environments or randomized strategies are merely allowed, not required. Solvers for stochastic games may incidentally produce randomized strategies to satisfy a functional specification (and some

H. Chockler and G. Weissenbacher (Eds.): CAV 2018, LNCS 10981, pp. 307–326, 2018.
https://doi.org/10.1007/978-3-319-96145-3_17

types of specification, e.g. multi-objective queries [4], may only be realizable by randomized strategies), but do not provide a general way to *enforce* randomness. Unlike most specifications used in reactive synthesis, our randomness requirement is a property of a system's *distribution* of behaviors, not of an individual behavior. While probabilistic specification languages like PCTL [12] can capture some such properties, the simple and natural randomness requirement we study here cannot be concisely expressed by existing languages (even those as powerful as SGL [2]). Thus, *randomized reactive synthesis* in our sense requires significantly different methods than those previously studied.

However, we argue that this type of synthesis is quite useful, because introducing randomness into the behavior of a system can often be beneficial, enhancing *variety*, *robustness*, and *unpredictability*. Example applications include:

- Synthesizing a black-box fuzz tester for a network service, we want a program that not only conforms to the protocol (perhaps only most of the time) but can generate many different sequences of packets: randomness ensures this.
- Synthesizing a controller for a robot exploring an unknown environment, randomness provides a low-memory way to increase coverage of the space. It can also help to reduce systematic bias in the exploration procedure.
- Synthesizing a controller for a patrolling surveillance robot, introducing randomness in planning makes the robot's future location harder to predict.

Adding randomness to a system in an *ad hoc* way could easily compromise its correctness. This paper shows how a randomness requirement can be integrated *into the synthesis process*, ensuring correctness as well as allowing trade-offs to be explored: how much randomness can be added while staying correct, or how strong can a specification be while admitting a desired amount of randomness?

To formalize randomized reactive synthesis we build on the idea of *control improvisation*, introduced in [6], formalized in [9], and further generalized in [8]. Control improvisation (CI) is the problem of constructing an *improviser*, a probabilistic algorithm which generates finite words subject to three constraints: a *hard constraint* that must always be satisfied, a *soft constraint* that need only be satisfied with some probability, and a *randomness constraint* that no word be generated with probability higher than a given bound. We define *reactive control improvisation* (RCI), where the improviser generates a word incrementally, alternating adding symbols with an adversarial environment. To perform synthesis in a finite window, we encode functional specifications and environment assumptions into the hard constraint, while the soft and randomness constraints allow us to tune how randomness is added to the system. The improviser obtained by solving the RCI problem is then a solution to the original synthesis problem.

The difficulty of solving reactive CI problems depends on the type of specification. We study several types commonly used in reactive synthesis, including reachability games (and variants, e.g. safety games) and formulas in the temporal logics LTL and LDL [5,18]. We also investigate the specification types studied in [8], showing how the complexity of the CI problem changes when adding reactivity. For every type of specification we obtain a randomized synthesis algorithm whose complexity matches that of ordinary reactive synthesis

(in a finite window). This suggests that reactive control improvisation should be feasible in applications like robotic task planning where reactive synthesis tools have proved effective.

In summary, the main contributions of this paper are:

- The reactive control improvisation (RCI) problem definition (Sect. 3);
- The notion of *width*, a quantitative generalization of "winning" game positions that measures *how many ways* a player can win from that position (Sect. 4);
- A characterization of when RCI problems are realizable in terms of width, and an explicit construction of an improviser (Sect. 4);
- A general method for constructing efficient improvisation schemes (Sect. 5);
- A polynomial-time improvisation scheme for reachability/safety games and deterministic finite automaton specifications (Sect. 6);
- PSPACE-hardness results for many other specification types including temporal logics, and matching polynomial-space improvisation schemes (Sect. 7).

Finally, Sect. 8 summarizes our results and gives directions for future work.

2 Background

2.1 Notation

Given an alphabet Σ, we write $|w|$ for the length of a finite word $w \in \Sigma^*$, λ for the empty word, Σ^n for the words of length n, and $\Sigma^{\leq n}$ for $\cup_{0 \leq i \leq n} \Sigma^i$, the set of all words of length at most n. We abbreviate deterministic/nondeterministic finite automaton by DFA/NFA, and context-free grammar by CFG. For an instance $\mathcal{X}$ of any such formalism, which we call a *specification*, we write $L(\mathcal{X})$ for the language (subset of Σ^*) it defines (note the distinction between a language and a representation thereof). We view formulas of Linear Temporal Logic (LTL) [18] and Linear Dynamic Logic (LDL) [5] as specifications using their natural semantics on finite words (see [5]).

We use the standard complexity classes #P and PSPACE, and the PSPACE-complete problem QBF of determining the truth of a quantified Boolean formula. For background on these classes and problems see for example [1].

Some specifications we use as examples are *reachability games* [16], where players' actions cause transitions in a state space and the goal is to reach a target state. We group these games, *safety games* where the goal is to *avoid* a set of states, and *reach-avoid* games combining reachability and safety goals [20], together as *reachability/safety games* (RSGs). We draw reachability games as graphs in the usual way: squares are adversary-controlled states, and states with a double border are target states.

2.2 Synthesis Games

Reactive control improvisation will be formalized in terms of a 2-player game which is essentially the standard *synthesis game* used in reactive synthesis [7]. However, our formulation is slightly different for compatibility with the definition of control improvisation, so we give a self-contained presentation here.

Fix a finite alphabet Σ. The players of the game will alternate picking symbols from Σ, building up a word. We can then specify the set of winning plays with a language over Σ. To simplify our presentation we assume that players strictly alternate turns and that any symbol from Σ is a legal move. These assumptions can be relaxed in the usual way by modifying the winning set appropriately.

Finite Words: While reactive synthesis is usually considered over infinite words, in this paper we focus on synthesis in a finite window, as it is unclear how best to generalize our randomness requirement to the infinite case. This assumption is not too restrictive, as solutions of bounded length are adequate for many applications. In fuzz testing, for example, we do not want to generate arbitrarily long files or sequences of packets. In robotic planning, we often want a plan that accomplishes a task within a certain amount of time. Furthermore, planning problems with liveness specifications can often be segmented into finite pieces: we do not need an infinite route for a patrolling robot, but can plan within a finite horizon and replan periodically. Replanning may even be *necessary* when environment assumptions become invalid. At any rate, we will see that the bounded case of reactive control improvisation is already highly nontrivial.

As a final simplification, we require that all plays have length exactly $n \in \mathbb{N}$. To allow a range $[m, n]$ we can simply add a new padding symbol to Σ and extend all shorter words to length n, modifying the winning set appropriately.

Definition 2.1. *A* history *h is an element of $\Sigma^{\leq n}$, representing the moves of the game played so far. We say the game has* ended *after h if $|h| = n$; otherwise it is* our turn *after h if $|h|$ is even, and* the adversary's turn *if $|h|$ is odd.*

Definition 2.2. *A* strategy *is a function $\sigma : \Sigma^{\leq n} \times \Sigma \to [0, 1]$ such that for any history $h \in \Sigma^{\leq n}$ with $|h| < n$, $\sigma(h, \cdot)$ is a probability distribution over Σ. We write $x \leftarrow \sigma(h)$ to indicate that x is a symbol randomly drawn from $\sigma(h, \cdot)$.*

Since strategies are randomized, fixing strategies for both players does not uniquely determine a play of the game, but defines a *distribution* over plays:

Definition 2.3. *Given a pair of strategies (σ, τ), we can generate a random* play *$\pi \in \Sigma^n$ as follows. Pick $\pi_0 \leftarrow \sigma(\lambda)$, then for i from 1 to $n-1$ pick $\pi_i \leftarrow \tau(\pi_0 \dots \pi_{i-1})$ if i is odd and $\pi_i \leftarrow \sigma(\pi_0 \dots \pi_{i-1})$ otherwise. Finally, put $\pi = \pi_0 \dots \pi_{n-1}$. We write $P_{\sigma,\tau}(\pi)$ for the probability of obtaining the play π. This extends to a* set *of plays $X \subseteq \Sigma^n$ in the natural way: $P_{\sigma,\tau}(X) = \sum_{\pi \in X} P_{\sigma,\tau}(\pi)$. Finally, the set of* possible *plays is $\Pi_{\sigma,\tau} = \{\pi \in \Sigma^n \mid P_{\sigma,\tau}(\pi) > 0\}$.*

The next definition is just the conditional probability of a play given a history, but works for histories with probability zero, simplifying our presentation.

Definition 2.4. *For any history* $h = h_0 \dots h_{k-1} \in \Sigma^{\leq n}$ *and word* $\rho \in \Sigma^{n-k}$*, we write* $P_{\sigma,\tau}(\rho|h)$ *for the probability that if we assign* $\pi_i = h_i$ *for* $i < k$ *and sample* $\pi_k, \dots, \pi_{n-1}$ *by the process above, then* $\pi_k \dots \pi_{n-1} = \rho$*.*

3 Problem Definition

3.1 Motivating Example

Consider synthesizing a planner for a surveillance drone operating near another, potentially adversarial drone. Discretizing the map into the 7×7 grid in Fig. 1 (ignoring the depicted trajectories for the moment), a route is a word over the four movement directions. Our specification is to visit the 4 circled locations in 30 moves without colliding with the adversary, assuming it cannot move into the 5 highlighted central locations.

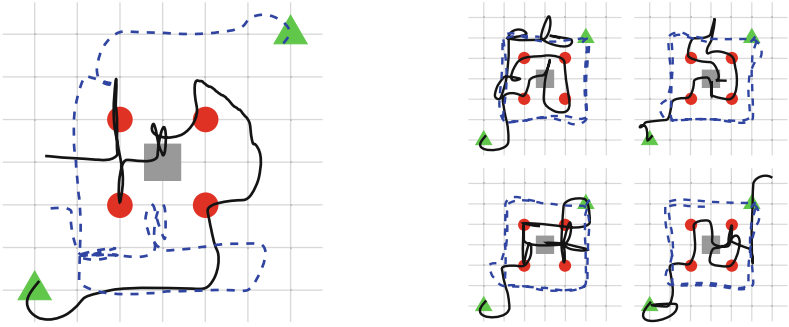

Fig. 1. Improvised trajectories for a patrolling drone (solid) avoiding an adversary (dashed). The adversary may not move into the circles or the square.

Existing reactive synthesis tools can produce a strategy for the patroller ensuring that the specification is always satisfied. However, the strategy may be deterministic, so that in response to a fixed adversary the patroller will always follow the same route. Then it is easy for a third party to predict the route, which could be undesirable, and is in fact unnecessary if there are many other ways the drone can satisfy its specification.

Reactive control improvisation addresses this problem by adding a new type of specification to the *hard constraint* above: a *randomness requirement* stating that no behavior should be generated with probability greater than a threshold ρ. If we set (say) $\rho = 1/5$, then any controller solving the synthesis problem must be able to satisfy the hard constraint in at least 5 different ways, never producing any given behavior more than 20% of the time. Our synthesis algorithm can in

fact compute the smallest ρ for which synthesis is possible, yielding a controller that is *maximally-randomized* in that the system's behavior is as close to a uniform distribution as possible.

To allow finer tuning of how randomness is introduced into the controller, our definition also includes a *soft constraint* which need only be satisfied with some probability $1-\epsilon$. This allows us to prefer certain safe behaviors over others. In our drone example, we require that with probability at least 3/4, we do not visit a circled location twice.

These hard, soft, and randomness constraints form an instance of our reactive control improvisation problem. Encoding the hard and soft constraints as DFAs, our algorithm (Sect. 6) produced a controller achieving the smallest realizable $\rho = 2.2 \times 10^{-12}$. We tested the controller using the PX4 autopilot [17] to refine the generated routes into control actions for a drone simulated in Gazebo [14] (videos and code are available online [11]). A selection of resulting trajectories are shown in Fig. 1 (the remainder in Appendix A of the full paper [10]): starting from the triangles, the patroller's path is solid, the adversary's dashed. The left run uses an adversary that moves towards the patroller when possible. The right runs, with a simple adversary moving in a fixed loop, illustrate the randomness of the synthesized controller.

3.2 Reactive Control Improvisation

Our formal notion of randomized reactive synthesis in a finite window is a reactive extension of *control improvisation* [8,9], which captures the three types of constraint (hard, soft, randomness) seen above. We use the notation of [8] for the specifications and languages defining the hard and soft constraints:

Definition 3.1 ([8]). *Given* hard *and* soft *specifications $\mathcal{H}$ and $\mathcal{S}$ of languages over Σ, an* improvisation *is a word $w \in L(\mathcal{H}) \cap \Sigma^n$. It is* admissible *if $w \in L(\mathcal{S})$. The set of all improvisations is denoted I, and admissible improvisations A.*

Running Example. We will use the following simple example throughout the paper: each player may increment $(+)$, decrement $(-)$, or leave unchanged $(=)$ a counter which is initially zero. The alphabet is $\Sigma = \{+, -, =\}$, and we set $n = 4$. The hard specification $\mathcal{H}$ is the DFA in Fig. 2 requiring that the counter stay within $[-2, 2]$. The soft specification $\mathcal{S}$ is a similar DFA requiring that the counter end at a nonnegative value.

Then for example the word $++==$ is an admissible improvisation, satisfying both hard and soft constraints, and so is in A. The word $+-=-$ on the other hand satisfies $\mathcal{H}$ but not $\mathcal{S}$, so it is in I but not A. Finally, $+++-$ does not satisfy $\mathcal{H}$, so it is not an improvisation at all and is not in I.

A reactive control improvisation problem is defined by $\mathcal{H}$, $\mathcal{S}$, and parameters ϵ and ρ. A solution is then a strategy which ensures that the hard, soft, and randomness constraints hold against every adversary. Formally, following [8,9]:

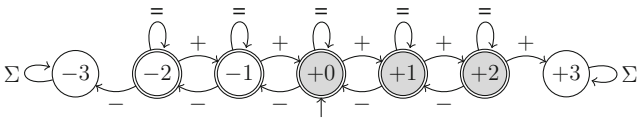

Fig. 2. The hard specification DFA $\mathcal{H}$ in our running example. The soft specification $\mathcal{S}$ is the same but with only the shaded states accepting.

Definition 3.2. *Given an* RCI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \rho)$ *with* $\mathcal{H}$*,* $\mathcal{S}$*, and* n *as above and* $\epsilon, \rho \in [0,1] \cap \mathbb{Q}$*, a strategy* σ *is an* improvising strategy *if it satisfies the following requirements for every adversary* τ*:*

Hard constraint: $P_{\sigma,\tau}(I) = 1$
Soft constraint: $P_{\sigma,\tau}(A) \geq 1 - \epsilon$
Randomness: $\forall \pi \in I,\ P_{\sigma,\tau}(\pi) \leq \rho$.

If there is an improvising strategy σ*, we say that* $\mathcal{C}$ *is* realizable*. An* improviser *for* $\mathcal{C}$ *is then an expected-finite time probabilistic algorithm implementing such a strategy* σ*, i.e. whose output distribution on input* $h \in \Sigma^{\leq n}$ *is* $\sigma(h, \cdot)$*.*

Definition 3.3. *Given an RCI instance* $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \rho)$*, the* reactive control improvisation *(RCI) problem is to decide whether* $\mathcal{C}$ *is realizable, and if so to generate an improviser for* $\mathcal{C}$*.*

Running Example. Suppose we set $\epsilon = 1/2$ and $\rho = 1/2$. Let σ be the strategy which picks $+$ or $-$ with equal probability in the first move, and thenceforth picks the action which moves the counter closest to ± 1 respectively. This satisfies the hard constraint, since if the adversary ever moves the counter to ± 2 we immediately move it back. The strategy also satisfies the soft constraint, since with probability $1/2$ we set the counter to $+1$ on the first move, and if the adversary moves to 0 we move back to $+1$ and remain nonnegative. Finally, σ also satisfies the randomness constraint, since each choice of first move happens with probability $1/2$ and so no play can be generated with higher probability. So σ is an improvising strategy and this RCI instance is realizable.

We will study classes of RCI problems with different types of specifications:

Definition 3.4. *If* HSpec *and* SSpec *are classes of specifications, then the class of RCI instances* $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \rho)$ *where* $\mathcal{H} \in$ HSpec *and* $\mathcal{S} \in$ SSpec *is denoted* RCI(HSpec, SSpec)*. We use the same notation for the decision problem associated with the class, i.e., given* $\mathcal{C} \in$ RCI(HSpec, SSpec)*, decide whether* $\mathcal{C}$ *is realizable. The* size $|\mathcal{C}|$ *of an RCI instance is the total size of the bit representations of its parameters, with* n *represented in unary and* ϵ, ρ *in binary.*

Finally, a *synthesis algorithm* in our context takes a specification in the form of an RCI instance and produces an implementation in the form of an improviser. This corresponds exactly to the notion of an improvisation scheme from [8]:

Definition 3.5 ([8]). *A* polynomial-time improvisation scheme *for a class $\mathcal{P}$ of RCI instances is an algorithm S with the following properties:*

Correctness: *For any $\mathcal{C} \in \mathcal{P}$, if $\mathcal{C}$ is realizable then $S(\mathcal{C})$ is an improviser for $\mathcal{C}$, and otherwise $S(\mathcal{C}) = \bot$.*

Scheme efficiency: *There is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that the runtime of S on any $\mathcal{C} \in \mathcal{P}$ is at most $p(|\mathcal{C}|)$.*

Improviser efficiency: *There is a polynomial $q : \mathbb{R} \to \mathbb{R}$ such that for every $\mathcal{C} \in \mathcal{P}$, if $G = S(\mathcal{C}) \neq \bot$ then G has expected runtime at most $q(|\mathcal{C}|)$.*

The first two requirements simply say that the scheme produces valid improvisers in polynomial time. The third is necessary to ensure that the improvisers themselves are efficient: otherwise, the scheme might for example produce improvisers running in time exponential in the size of the specification.

A main goal of our paper is to determine for which types of specifications there exist polynomial-time improvisation schemes. While we do find such algorithms for important classes of specifications, we will also see that determining the realizability of an RCI instance is often PSPACE-hard. Therefore we also consider *polynomial-space improvisation schemes*, defined as above but replacing time with space.

4 Existence of Improvisers

4.1 Width and Realizability

The most basic question in reactive synthesis is whether a specification is realizable. In *randomized* reactive synthesis, the question is more delicate because the randomness requirement means that it is no longer enough to ensure some property regardless of what the adversary does: there must be *many ways* to do so. Specifically, there must be at least $1/\rho$ improvisations if we are to generate each of them with probability at most ρ. Furthermore, at least this many improvisations must be *possible* given an unknown adversary: even if many exist, the adversary may be able to force us to use only a single one. We introduce a new notion of the size of a set of plays that takes this into account.

Definition 4.1. *The* width *of $X \subseteq \Sigma^n$ is $W(X) = \max_\sigma \min_\tau |X \cap \Pi_{\sigma,\tau}|$.*

The width counts how many distinct plays can be generated regardless of what the adversary does. Intuitively, a "narrow" game—one whose set of winning plays has small width—is one in which the adversary can force us to choose among only a few winning plays, while in a "wide" one we always have many safe choices available. Note that *which* particular plays can be generated depends on the adversary: the width only measures *how many* can be generated. For example, $W(X) = 1$ means that a play in X can always be generated, but possibly a different element of X for different adversaries.

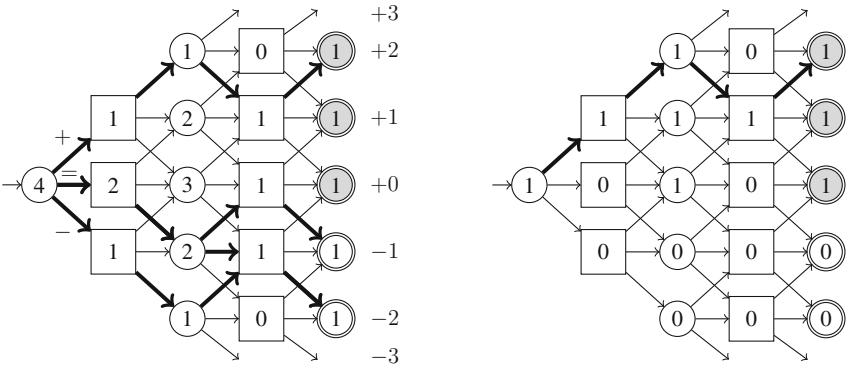

Fig. 3. Synthesis game for our running example. States are labeled with the widths of I (left) and A (right) given a history ending at that state.

Running Example. Figure 3 shows the synthesis game for our running example: paths ending in circled or shaded states are plays in I or A respectively (ignore the state labels for now). At left, the bold arrows show the 4 plays in I possible against the adversary that moves away from 0, and down at 0. This shows $W(I) \leq 4$, and in fact 4 plays are possible against any adversary, so $W(I) = 4$. Similarly, at right we see that $W(A) = 1$.

It will be useful later to have a *relative* version of width that counts how many plays are possible *from a given position*:

Definition 4.2. *Given a set of plays $X \subseteq \Sigma^n$ and a history $h \in \Sigma^{\leq n}$, the* width of X given h *is* $W(X|h) = \max_\sigma \min_\tau |\{\pi \mid h\pi \in X \wedge P_{\sigma,\tau}(\pi|h) > 0\}|$.

This is a direct generalization of "winning" positions: if X is the set of winning plays, then $W(X|h)$ counts the number of ways to win from h.

We will often use the following basic properties of $W(X|h)$ without comment (for lack of space this proof and the details of later proof sketches are deferred to Appendix B of the full paper [10]). Note that (3)–(5) provide a recursive way to compute widths that we will use later, and which is illustrated by the state labels in Fig. 3.

Lemma 4.1. *For any set of plays $X \subseteq \Sigma^n$ and history $h \in \Sigma^{\leq n}$:*

1. $0 \leq W(X|h) \leq |\Sigma|^{n-|h|}$;
2. $W(X|\lambda) = W(X)$;
3. *if* $|h| = n$, *then* $W(X|h) = \mathbb{1}_{h \in X}$;
4. *if it is our turn after* h, *then* $W(X|h) = \sum_{u \in \Sigma} W(X|hu)$;
5. *if it is the adversary's turn after* h, *then* $W(X|h) = \min_{u \in \Sigma} W(X|hu)$.

Now we can state the realizability conditions, which are simply that I and A have sufficiently large width. In fact, the conditions turn out to be exactly the same as those for non-reactive CI except that width takes the place of size [9].

Theorem 4.1. *The following are equivalent:*

(1) $\mathcal{C}$ *is realizable.*
(2) $W(I) \geq 1/\rho$ *and* $W(A) \geq (1-\epsilon)/\rho$.
(3) There is an improviser for $\mathcal{C}$.

Running Example. We saw above that our example was realizable with $\epsilon = \rho = 1/2$, and indeed $4 = W(I) \geq 1/\rho = 2$ and $1 = W(A) \geq (1-\epsilon)/\rho = 1$. However, if we put $\rho = 1/3$ we violate the second inequality and the instance is not realizable: essentially, we need to distribute probability $1 - \epsilon = 1/2$ among plays in A (to satisfy the soft constraint), but since $W(A) = 1$, against some adversaries we can only generate one play in A and would have to give it the whole $1/2$ (violating the randomness requirement).

The difficult part of the Theorem is constructing an improviser when the inequalities (2) hold. Despite the similarity in these conditions to the non-reactive case, the construction is much more involved. We begin with a general overview.

4.2 Improviser Construction: Discussion

Our improviser can be viewed as an extension of the classical random-walk reduction of uniform sampling to counting [21]. In that algorithm (which was used in a similar way for DFA specifications in [8,9]), a uniform distribution over paths in a DAG is obtained by moving to the next vertex with probability proportional to the number of paths originating at it. In our case, which plays are possible depends on the adversary, but the width still tells us *how many* plays are possible. So we could try a random walk using widths as weights: e.g. on the first turn in Fig. 3, picking $+$, $-$, and $=$ with probabilities $1/4$, $2/4$, and $1/4$ respectively. Against the adversary shown in Fig. 3, this would indeed yield a uniform distribution over the four possible plays in I.

However, the soft constraint may require a non-uniform distribution. In the running example with $\epsilon = \rho = 1/2$, we need to generate the single possible play in A with probability $1/2$, not just the uniform probability $1/4$. This is easily fixed by doing the random walk with a *weighted average* of the widths of I and A: specifically, move to position h with probability proportional to $\alpha W(A|h) + \beta(W(I|h) - W(A|h))$. In the example, this would result in plays in A getting probability α and those in $I \setminus A$ getting probability β. Taking α sufficiently large, we can ensure the soft constraint is satisfied.

Unfortunately, this strategy can fail if the adversary makes *more* plays available than the width guarantees. Consider the game on the left of Fig. 4, where $W(I) = 3$ and $W(A) = 2$. This is realizable with $\epsilon = \rho = 1/3$, but no values of α and β yield improvising strategies, essentially because an adversary moving from X to Z breaks the worst-case assumption that the adversary will minimize the number of possible plays by moving to Y. In fact, this instance is realizable but not by any memoryless strategy. To see this, note that all such strategies can be parametrized by the probabilities p and q in Fig. 4. To satisfy the randomness

Fig. 4. Reachability games where a naïve random walk, and all memoryless strategies, fail (left) and where no strategy can optimize either ϵ or ρ against every adversary simultaneously (right).

constraint against the adversary that moves from X to Y, both p and $(1-p)q$ must be at most $1/3$. To satisfy the soft constraint against the adversary that moves from X to Z we must have $pq + (1-p)q \geq 2/3$, so $q \geq 2/3$. But then $(1-p)q \geq (1-1/3)(2/3) = 4/9 > 1/3$, a contradiction.

To fix this problem, our improvising strategy $\hat{\sigma}$ (which we will fully specify in Algorithm 1 below) takes a simplistic approach: it tracks how many plays in A and I are expected to be possible based on their widths, and if more are available it ignores them. For example, entering state Z from X there are 2 ways to produce a play in I, but since $W(I|X) = 1$ we ignore the play in $I \setminus A$. Extra plays in A are similarly ignored by being treated as members of $I \setminus A$. Ignoring unneeded plays may seem wasteful, but the proof of Theorem 4.1 will show that $\hat{\sigma}$ nevertheless achieves the best possible ϵ:

Corollary 4.1. *$\mathcal{C}$ is realizable iff $W(I) \geq 1/\rho$ and $\epsilon \geq \epsilon_{\text{opt}} \equiv \max(1 - \rho W(A), 0)$. Against any adversary, the error probability of Algorithm 1 is at most ϵ_{opt}.*

Thus, if *any* improviser can achieve an error probability ϵ, ours does. We could ask for a stronger property, namely that against each adversary the improviser achieves the smallest possible error probability *for that adversary*. Unfortunately, this is impossible in general. Consider the game on the right in Fig. 4, with $\rho = 1$. Against the adversary which always moves up, we can achieve $\epsilon = 0$ with the strategy that at P moves to Q. We can also achieve $\epsilon = 0$ against the adversary that always moves down, but only with a *different* strategy, namely the one that at P moves to R. So there is no single strategy that achieves the optimal ϵ for every adversary. A similar argument shows that there is also no strategy achieving the smallest possible ρ for every adversary. In essence, optimizing ϵ or ρ in every case would require the strategy to depend on the adversary.

4.3 Improviser Construction: Details

Our improvising strategy, as outlined in the previous section, is shown in Algorithm 1. We first compute α and β, the (maximum) probabilities for generating elements of A and $I \setminus A$ respectively. As in [8], we take α as large as possible given $\alpha \leq \rho$, and determine β from the probability left over (modulo a couple corner cases).

Algorithm 1. the strategy $\hat{\sigma}$

$\alpha \leftarrow \min(\rho, 1/W(A))$ (or 0 instead if $W(A) = 0$)
$\beta \leftarrow (1 - \alpha W(A))/(W(I) - W(A))$ (or 0 instead if $W(I) - W(A) = 0$)
$m^A \leftarrow W(A)$, $m^I \leftarrow W(I)$
$h \leftarrow \lambda$
while the game is not over after h **do**
  if it is our turn after h **then**
    $m^A_u, m^I_u \leftarrow$ PARTITION(m^A, m^I, h) ▷ returns values for each $u \in \Sigma$
    for each $u \in \Sigma$, put $t_u \leftarrow \alpha m^A_u + \beta(m^I_u - m^A_u)$
    pick $u \in \Sigma$ with probability proportional to t_u and append it to h
    $m^A \leftarrow m^A_u$, $m^I \leftarrow m^I_u$
  else
    the adversary picks $u \in \Sigma$ given the history h; append it to h
return h

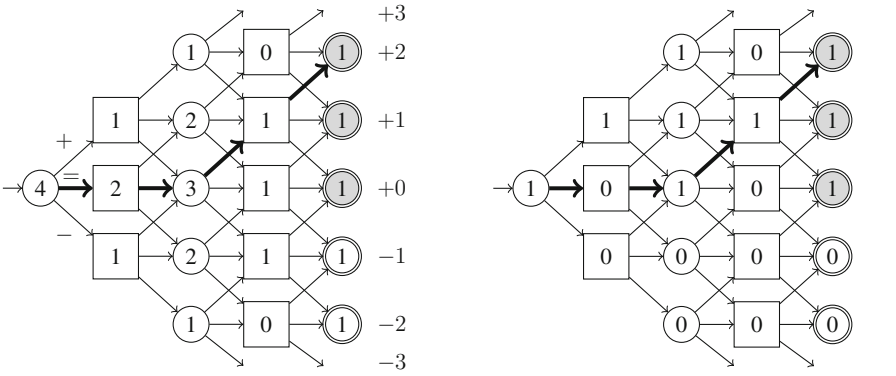

Fig. 5. A run of Algorithm 1, labeling states with corresponding widths of I (left) and A (right).

Next we initialize m^A and m^I, our expectations for how many plays in A and I respectively are still possible to generate. Initially these are given by $W(A)$ and $W(I)$, but as we saw above it is possible for more plays to become available. The function PARTITION handles this, deciding which m^A (resp., m^I) out of the available $W(A|h)$ ($W(I|h)$) plays we will use. The behavior of PARTITION is defined by the following lemma; its proof (in Appendix B [10]) greedily takes the first m^A possible plays in A under some canonical order and the first $m^I - m^A$ of the remaining plays in I.

Lemma 4.2. *If it is our turn after $h \in \Sigma^{\leq n}$, and $m^A, m^I \in \mathbb{Z}$ satisfy $0 \leq m^A \leq m^I \leq W(I|h)$ and $m^A \leq W(A|h)$, there are integer partitions $\sum_{u\in\Sigma} m^A_u$ and $\sum_{u\in\Sigma} m^I_u$ of m^A and m^I respectively such that $0 \leq m^A_u \leq m^I_u \leq W(I|hu)$ and $m^A_u \leq W(A|hu)$ for all $u \in \Sigma$. These are computable in poly-time given oracles for $W(I|\cdot)$ and $W(A|\cdot)$.*

Finally, we perform the random walk, moving from position h to hu with (unnormalized) probability t_u, the weighted average described above.

Running Example. With $\epsilon = \rho = 1/2$, as before $W(A) = 1$ and $W(I) = 4$ so $\alpha = 1/2$ and $\beta = 1/6$. On the first move, m^A and m^I match $W(A|h)$ and $W(I|h)$, so all plays are used and PARTITION returns $(W(A|hu), W(I|hu))$ for each $u \in \Sigma$. Looking up these values in Fig. 5, we see $(m^A_=, m^I_=) = (0, 2)$ and so $t(=) = 2\beta = 1/3$. Similarly $t(+) = \alpha = 1/2$ and $t(-) = \beta = 1/6$. We choose an action according to these weights; suppose $=$, so that we update $m^A \leftarrow 0$ and $m^I \leftarrow 2$, and suppose the adversary responds with $=$. From Fig. 5, $W(A| ==) = 1$ and $W(I| ==) = 3$, whereas $m^A = 0$ and $m^I = 2$. So PARTITION discards a play, say returning $(m^A_u, m^I_u) = (0, 1)$ for $u \in \{+, =\}$ and $(0, 0)$ for $u \in \{-\}$. Then $t(+) = t(=) = \beta = 1/6$ and $t(-) = 0$. So we pick $+$ or $=$ with equal probability, say $+$. If the adversary responds with $+$, we get the play $==++$, shown in bold on Fig. 5. As desired, it satisfies the hard constraint.

The next few lemmas establish that $\hat{\sigma}$ is well-defined and in fact an improvising strategy, allowing us to prove Theorem 4.1. Throughout, we write $m^A(h)$ (resp., $m^I(h)$) for the value of m^A (m^I) at the start of the iteration for history h. We also write $t(h) = \alpha m^A(h) + \beta(m^I(h) - m^A(h))$ (so $t(hu) = t_u$ when we pick u).

Lemma 4.3. *If $W(I) \geq 1/\rho$, then $\hat{\sigma}$ is a well-defined strategy and $P_{\hat{\sigma},\tau}(I) = 1$ for every adversary τ.*

Proof (sketch). An easy induction on h shows the conditions of Lemma 4.2 are always satisfied, and that $t(h)$ is always positive since we never pick a u with $t_u = 0$. So $\sum_u t_u = t(h) > 0$ and $\hat{\sigma}$ is well-defined. Furthermore, $t(h) > 0$ implies $m^I(h) > 0$, so for any $h \in \Pi_{\hat{\sigma},\tau}$ we have $\mathbb{1}_{h \in I} = W(I|h) \geq m^I(h) > 0$ and thus $h \in I$. □

Lemma 4.4. *If $W(I) \geq 1/\rho$, then $P_{\hat{\sigma},\tau}(A) \geq \min(\rho W(A), 1)$ for every τ.*

Proof (sketch). Because of the $\alpha m^A(h)$ term in the weights $t(h)$, the probability of obtaining a play in A starting from h is at least $\alpha m^A(h)/t(h)$ (as can be seen by induction on h in order of decreasing length). Then since $m^A(\lambda) = W(A)$ and $t(\lambda) = 1$ we have $P_{\hat{\sigma},\tau}(A) \geq \alpha W(A) = \min(\rho W(A), 1)$. □

Lemma 4.5. *If $W(I) \geq 1/\rho$, then $P_{\hat{\sigma},\tau}(\pi) \leq \rho$ for every $\pi \in \Sigma^n$ and τ.*

Proof (sketch). If the adversary is deterministic, the weights we use for our random walk yield a distribution where each play π has probability either α or β (depending on whether $m^A(\pi) = 1$ or 0). If the adversary assigns nonzero probability to multiple choices this only decreases the probability of individual plays. Finally, since $W(I) \geq 1/\rho$ we have $\alpha, \beta \leq \rho$. □

Proof (of Theorem 4.1). We use a similar argument to that of [8].

(1)⇒(2) Suppose σ is an improvising strategy, and fix any adversary τ. Then $\rho|\Pi_{\sigma,\tau} \cap I| = \sum_{\pi \in \Pi_{\sigma,\tau} \cap I} \rho \geq \sum_{\pi \in I} P_{\sigma,\tau}(\pi) = P_{\sigma,\tau}(I) = 1$, so $|\Pi_{\sigma,\tau} \cap I| \geq 1/\rho$. Since τ is arbitrary, this implies $W(I) \geq 1/\rho$. Since $A \subseteq I$, we also have $\rho|\Pi_{\sigma,\tau} \cap A| = \sum_{\pi \in \Pi_{\sigma,\tau} \cap A} \rho \geq \sum_{\pi \in A} P_{\sigma,\tau}(\pi) = P_{\sigma,\tau}(A) \geq 1 - \epsilon$, so $|\Pi_{\sigma,\tau} \cap A| \geq (1-\epsilon)/\rho$ and thus $W(A) \geq (1-\epsilon)/\rho$.

(2)⇒(3) By Lemmas 4.3 and 4.5, $\hat{\sigma}$ is well-defined and satisfies the hard and randomness constraints. By Lemma 4.4, $P_{\hat{\sigma},\tau}(A) \geq \min(\rho W(A), 1) \geq 1 - \epsilon$, so $\hat{\sigma}$ also satisfies the soft constraint and thus is an improvising strategy. Its transition probabilities are rational, so it can be implemented by an expected finite-time probabilistic algorithm, which is then an improviser for $\mathcal{C}$.

(3)⇒(1) Immediate. □

Proof (of Corollary 4.1). The inequalities in the statement are equivalent to those of Theorem 4.1 (2). By Lemma 4.4, we have $P_{\hat{\sigma},\tau}(A) \geq \min(\rho W(A), 1)$. So the error probability is at most $1 - \min(\rho W(A), 1) = \epsilon_{\text{opt}}$. □

5 A Generic Improviser

We now use the construction of Sect. 4 to develop a generic improvisation scheme usable with any class of specifications Spec supporting the following operations:

Intersection: Given specs $\mathcal{X}$ and $\mathcal{Y}$, find $\mathcal{Z}$ such that $L(\mathcal{Z}) = L(\mathcal{X}) \cap L(\mathcal{Y})$.

Width Measurement: Given a specification $\mathcal{X}$, a length $n \in \mathbb{N}$ in unary, and a history $h \in \Sigma^{\leq n}$, compute $W(X|h)$ where $X = L(\mathcal{X}) \cap \Sigma^n$.

Efficient algorithms for these operations lead to efficient improvisation schemes:

Theorem 5.1. *If the operations on* Spec *above take polynomial time (resp. space), then* RCI(Spec, Spec) *has a polynomial-time (space) improvisation scheme.*

Proof. Given an instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \rho)$ in RCI(Spec, Spec), we first apply intersection to $\mathcal{H}$ and $\mathcal{S}$ to obtain $\mathcal{A} \in$ Spec such that $L(\mathcal{A}) \cap \Sigma^n = A$. Since intersection takes polynomial time (space), $\mathcal{A}$ has size polynomial in $|\mathcal{C}|$. Next we use width measurement to compute $W(I) = W(L(\mathcal{H}) \cap \Sigma^n|\lambda)$ and $W(A) = W(L(\mathcal{A}) \cap \Sigma^n|\lambda)$. If these violate the inequalities in Theorem 4.1, then $\mathcal{C}$ is not realizable and we return $\perp$. Otherwise $\mathcal{C}$ is realizable, and $\hat{\sigma}$ above is an improvising strategy. Furthermore, we can construct an expected finite-time probabilistic algorithm implementing $\hat{\sigma}$, using width measurement to instantiate the oracles needed by Lemma 4.2. Determining $m^A(h)$ and $m^I(h)$ takes $O(n)$ invocations of Partition, each of which is poly-time relative to the width measurements. These take time (space) polynomial in $|\mathcal{C}|$, since $\mathcal{H}$ and $\mathcal{A}$ have size polynomial in $|\mathcal{C}|$. As $m^A, m^I \leq |\Sigma|^n$, they have polynomial bitwidth and so the arithmetic required to compute t_u for each $u \in \Sigma$ takes polynomial time. Therefore the total expected runtime (space) of the improviser is polynomial. □

Note that as a byproduct of testing the inequalities in Theorem 4.1, our algorithm can compute the best possible error probability ϵ_{opt} given $\mathcal{H}$, $\mathcal{S}$, and ρ (see Corollary 4.1). Alternatively, given ϵ, we can compute the best possible ρ.

We will see below how to efficiently compute widths for DFAs, so Theorem 5.1 yields a polynomial-time improvisation scheme. If we allow polynomial-*space* schemes, we can use a general technique for width measurement that only requires a very weak assumption on the specifications, namely testability in polynomial space:

Theorem 5.2. $\mathrm{RCI}(\mathrm{PSA}, \mathrm{PSA})$ *has a polynomial-space improvisation scheme, where* PSA *is the class of polynomial-space decision algorithms.*

Proof (sketch). We apply Theorem 5.1, computing widths recursively using Lemmas 4.1, (3)–(5). As in the PSPACE QBF algorithm, the current path in the recursive tree and required auxiliary storage need only polynomial space. □

6 Reachability Games and DFAs

Now we develop a polynomial-time improvisation scheme for RCI instances with DFA specifications. This also provides a scheme for reachability/safety games, whose winning conditions can be straightforwardly encoded as DFAs.

Suppose D is a DFA with states V, accepting states T, and transition function $\delta : V \times \Sigma \to V$. Our scheme is based on the fact that $W(L(D)|h)$ depends only on the state of D reached on input h, allowing these widths to be computed by dynamic programming. Specifically, for all $v \in V$ and $i \in \{0, \ldots, n\}$ we define:

$$C(v,i) = \begin{cases} \mathbb{1}_{v \in T} & i = n \\ \min_{u \in \Sigma} C(\delta(v,u), i+1) & i < n \wedge i \text{ odd} \\ \sum_{u \in \Sigma} C(\delta(v,u), i+1) & \text{otherwise.} \end{cases}$$

Running Example. Figure 6 shows the values $C(v,i)$ in rows from $i = n$ downward. For example, $i = 2$ is our turn, so $C(1,2) = C(0,3) + C(1,3) + C(2,3) = 1+1+0 = 2$, while $i = 3$ is the adversary's turn, so $C(-3,3) = \min\{C(-3,4)\} = \min\{0\} = 0$. Note that the values in Fig. 6 agree with the widths $W(I|h)$ shown in Fig. 5.

Lemma 6.1. *For any history $h \in \Sigma^{\leq n}$, writing $X = L(D) \cap \Sigma^n$ we have $W(X|h) = C(D(h), |h|)$, where $D(h)$ is the state reached by running D on h.*

Proof. We prove this by induction on $i = |h|$ in decreasing order. In the base case $i = n$, we have $W(X|h) = \mathbb{1}_{h \in X} = \mathbb{1}_{D(h) \in T} = C(D(h), n)$. Now take any history $h \in \Sigma^{\leq n}$ with $|h| = i < n$. By hypothesis, for any $u \in \Sigma$ we have $W(X|hu) = C(D(hu), i+1)$. If it is our turn after h, then $W(X|h) = \sum_{u \in \Sigma} W(X|hu) = \sum_{u \in \Sigma} C(D(hu), i+1) = C(D(h), i)$ as desired. If instead it is the adversary's turn after h, then $W(X|h) = \min_{u \in \Sigma} W(X|hu) = \min_{u \in \Sigma} C(D(hu), i+1) = C(D(h), i)$ again as desired. So by induction the hypothesis holds for any i. □

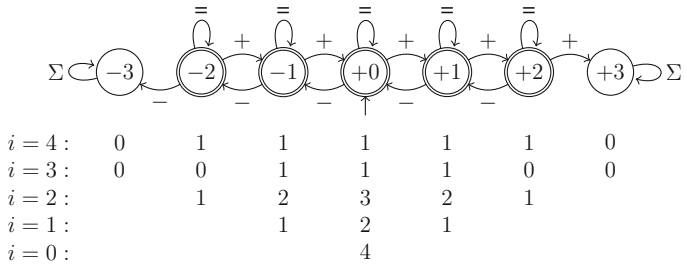

Fig. 6. The hard specification DFA $\mathcal{H}$ in our running example, showing how $W(I|h)$ is computed.

Theorem 6.1. RCI (DFA, DFA) *has a polynomial-time improvisation scheme.*

Proof. We implement Theorem 5.1. Intersection can be done with the standard product construction. For width measurement we compute the quantities $C(v, i)$ by dynamic programming (from $i = n$ down to $i = 0$) and apply Lemma 6.1. □

7 Temporal Logics and Other Specifications

In this section we analyze the complexity of reactive control improvisation for specifications in the popular temporal logics LTL and LDL. We also look at NFA and CFG specifications, previously studied for non-reactive CI [8], to see how their complexities change in the reactive case.

For LTL specifications, reactive control improvisation is PSPACE-hard because this is already true of ordinary reactive synthesis in a finite window (we suspect this has been observed but could not find a proof in the literature).

Theorem 7.1. *Finite-window reactive synthesis for* LTL *is* PSPACE*-hard.*

Proof (sketch). Given a *QBF* $\phi = \exists x \forall y \ldots \chi$, we can view assignments to its variables as traces over a single proposition. In polynomial time we can construct an LTL formula ψ whose models are the satisfying assignments of χ. Then there is a winning strategy to generate a play satisfying ψ iff ϕ is true. □

Corollary 7.1. RCI (LTL, Σ^*) *and* RCI (Σ^*, LTL) *are* PSPACE*-hard.*

This is perhaps disappointing, but is an inevitable consequence of LTL subsuming Boolean formulas. On the other hand, our general polynomial-space scheme applies to LTL and its much more expressive generalization LDL:

Theorem 7.2. RCI (LDL, LDL) *has a polynomial-space improvisation scheme.*

Proof. This follows from Theorem 5.2, since satisfaction of an LDL formula by a finite word can be checked in polynomial time (e.g. by combining dynamic programming on subformulas with a regular expression parser). □

Thus for temporal logics polynomial-time algorithms are unlikely, but adding randomization to reactive synthesis does not increase its complexity.

The same is true for NFA and CFG specifications, where it is again PSPACE-hard to find even a single winning strategy:

Theorem 7.3. *Finite-window reactive synthesis for* NFA*s is* PSPACE*-hard.*

Proof (sketch). Reduce from QBF as in Theorem 7.1, constructing an NFA accepting the satisfying assignments of χ (as done in [13]). □

Corollary 7.2. $\mathrm{RCI}(\mathrm{NFA}, \Sigma^*)$ *and* $\mathrm{RCI}(\Sigma^*, \mathrm{NFA})$ *are* PSPACE*-hard.*

Theorem 7.4. $\mathrm{RCI}(\mathrm{CFG}, \mathrm{CFG})$ *has a polynomial-space improvisation scheme.*

Proof. By Theorem 5.2, since CFG parsing can be done in polynomial time. □

Since NFAs can be converted to CFGs in polynomial time, this completes the picture for the kinds of CI specifications previously studied. In non-reactive CI, DFA specifications admit a polynomial-time improvisation scheme while for NFAs/CFGs the CI problem is #P-equivalent [8]. Adding reactivity, DFA specifications remain polynomial-time while NFAs and CFGs move up to PSPACE.

Table 1. Complexity of the reactive control improvisation problem for various types of hard and soft specifications $\mathcal{H}$, $\mathcal{S}$. Here PSPACE indicates that checking realizability is PSPACE-hard, and that there is a polynomial-space improvisation scheme.

$\mathcal{H}\backslash\mathcal{S}$	**RSG**	**DFA**	**NFA**	**CFG**	**LTL**	**LDL**
RSG	poly-time		PSPACE			
DFA						
NFA						
CFG						
LTL						
LDL						

8 Conclusion

In this paper we introduced *reactive control improvisation* as a framework for modeling reactive synthesis problems where random but controlled behavior is desired. RCI provides a natural way to tune the amount of randomness while ensuring that safety or other constraints remain satisfied. We showed that RCI problems can be efficiently solved in many cases occurring in practice, giving a polynomial-time improvisation scheme for reachability/safety or DFA specifications. We also showed that RCI problems with specifications in LTL or LDL, popularly used in planning, have the PSPACE-hardness typical of bounded games,

and gave a matching polynomial-space improvisation scheme. This scheme generalizes to any specification checkable in polynomial space, including NFAs, CFGs, and many more expressive formalisms. Table 1 summarizes these results.

These results show that, at a high level, finding a maximally-randomized strategy using RCI is no harder than finding any winning strategy at all: for specifications yielding games solvable in polynomial time (respectively, space), we gave polynomial-time (space) improvisation schemes. We therefore hope that in applications where ordinary reactive synthesis has proved tractable, our notion of randomized reactive synthesis will also. In particular, we expect our DFA scheme to be quite practical, and are experimenting with applications in robotic planning. On the other hand, our scheme for temporal logic specifications seems unlikely to be useful in practice without further refinement. An interesting direction for future work would be to see if modern solvers for quantified Boolean formulas (QBF) could be leveraged or extended to solve these RCI problems. This could be useful even for DFA specifications, as conjoining many simple properties can lead to exponentially-large automata. Symbolic methods based on constraint solvers would avoid such blow-up.

We are also interested in extending the RCI problem definition to unbounded or infinite words, as typically used in reactive synthesis. These extensions, as well as that to continuous signals, would be useful in robotic planning, cyber-physical system testing, and other applications. However, it is unclear how best to adapt our randomness constraint to settings where the improviser can generate infinitely many words. In such settings the improviser could assign arbitrarily small or even zero probability to every word, rendering the randomness constraint trivial. Even in the bounded case, RCI extensions with more complex randomness constraints than a simple upper bound on individual word probabilities would be worthy of study. One possibility would be to more directly control diversity and/or unpredictability by requiring the distribution of the improviser's output to be close to uniform after transformation by a given function.

Acknowledgements. The authors would like to thank Markus Rabe, Moshe Vardi, and several anonymous reviewers for helpful discussions and comments, and Ankush Desai and Tommaso Dreossi for assistance with the drone simulations. This work is supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1106400, by NSF grants CCF-1139138 and CNS-1646208, by DARPA under agreement number FA8750-16-C0043, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, New York (2009)
2. Baier, C., Brázdil, T., Größer, M., Kučera, A.: Stochastic game logic. Acta Inf. **49**(4), 203–224 (2012)

3. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: hardware from PSL. Electron. Notes Theor. Comput. Sci. **190**, 3–16 (2007). Proceedings of the 6th International Workshop on Compiler Optimization Meets Compiler Verification (COCV 2007). http://www.sciencedirect.com/science/article/pii/S157106610700583X
4. Chen, T., Forejt, V., Kwiatkowska, M., Simaitis, A., Wiltsche, C.: On stochastic games with multiple objectives. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 266–277. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40313-2_25
5. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence. IJCAI 2013, pp. 854–860. AAAI Press (2013). http://dl.acm.org/citation.cfm?id=2540128.2540252
6. Donze, A., Libkind, S., Seshia, S.A., Wessel, D.: Control improvisation with application to music. Technical reports UCB/EECS-2013-183, EECS Department, University of California, Berkeley, Nov 2013. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-183.html
7. Finkbeiner, B.: Synthesis of reactive systems. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering. NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press, Amsterdam (2016)
8. Fremont, D.J., Donzé, A., Seshia, S.A.: Control improvisation. arXiv preprint (2017)
9. Fremont, D.J., Donzé, A., Seshia, S.A., Wessel, D.: Control improvisation. In: 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 463–474 (2015)
10. Fremont, D.J., Seshia, S.A.: Reactive control improvisation. arXiv preprint (2018)
11. Fremont, D.J., Seshia, S.A.: Reactive control improvisation website (2018). https://math.berkeley.edu/~dfremont/reactive.html
12. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comput. **6**(5), 512–535 (1994)
13. Kannan, S., Sweedyk, Z., Mahaney, S.: Counting and random generation of strings in regular languages. In: 6th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 551–557. SIAM (1995)
14. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 3, pp. 2149–2154. IEEE (2004)
15. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. Rob. **25**(6), 1370–1381 (2009)
16. Mazala, R.: Infinite games. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata Logics, and Infinite Games. LNCS, vol. 2500, pp. 23–38. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36387-4_2
17. Meier, L., Honegger, D., Pollefeys, M.: PX4: a node-based multithreaded open source robotics framework for deeply embedded platforms. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 6235–6240. IEEE (2015)
18. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), pp. 46–57. IEEE (1977)
19. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming

Languages. POPL 1989, pp. 179–190. ACM, New York (1989). http://doi.acm.org/10.1145/75277.75293
20. Tomlin, C., Lygeros, J., Sastry, S.: Computing controllers for nonlinear hybrid systems. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 238–255. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48983-5_22
21. Wilf, H.S.: A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. Adv. Math. **24**(2), 281–291 (1977)