On Learning and Testing Dynamic Environments*

Oded Goldreich† Dana Ron‡

March 21, 2016

Abstract

We initiate a study of learning and testing dynamic environments, focusing on environments that evolve according to a fixed local rule. The (proper) learning task consists of obtaining the initial configuration of the environment, whereas for non-proper learning it suffices to predict its future values. The testing task consists of checking whether the environment has indeed evolved from some initial configuration according to the known evolution rule. We focus on the temporal aspect of these computational problems, which is reflected in two requirements: (1) it is not possible to "go back to the past" and make a query concerning the environment at time t after having made a query concerning time $t' > t$; and (2) only a small portion of the environment is inspected in each time unit.

We present several general results, extensive studies of two special cases, and a host of open problems. The general results illustrate the significance of the temporal aspect of the current model (i.e., the difference between the current model and the standard model) as well as the preservation of some relations that hold in the standard model. The two special cases that we study are linear rules of evolution and rules of evolution that represent simple movement of objects. Specifically, we show that evolution according to any linear rule can be tested within a total number of queries that is sublinear in the size of the environment, and that evolution according to a simple one-dimensional movement rule can be tested within a total number of queries that is independent of the size of the environment.

Keywords: Multi-dimensional cellular automata, Property Testing, PAC Learning, one-sided versus two-sided error probability, non-adaptivity, Locally Testable Codes, One-Way Functions,

*This research was partially supported by the Israel Science Foundation (grant No. 671/13). An extended abstract has appeared in the proceedings of the *55th FOCS*, 2014. A full preliminary version was posted on *ECCC*, as TR14-029, in March 2014.

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. oded.goldreich@weizmann.ac.il

‡Department of EE–Systems, Tel-Aviv University, Ramat-Aviv, Israel. danar@eng.tau.ac.il

Contents

1 Introduction

We initiate a study of sublinear algorithms for testing and learning *dynamic environments that evolve according to a local rule*. That is, the content of the environment at each time and in each location is determined by the contents of the local neighborhood of that location at the previous time.

One archetypical example of such environments is that of a collection of small components of a large system that interact at a local level (i.e., each component may change its local state based on the state of its neighbors). Indeed, the model of (two-dimensional) cellular automata was invented and studied as a model for such environments, and one may view our study as a study of sublinear algorithms for testing and learning the evolution of cellular automata. Another archetypical example is that of a collection of objects that move in (three-dimensional) space such that their movements may be affected by collisions (or near collisions) with other objects. Such a motion can also be represented as an evolution of a (three-dimensional) cellular automaton.

The *sublinear aspect* of our model is reflected in the postulate that the algorithm can only probe a small portion of the environment at each time unit, where the environment evolves in time (and is thus potentially different in each time unit). Yet, as stated above, the evolution of the environment is not arbitrary, but is rather based on local rules.

1.1 The basic model

The environment is viewed as a d-dimensional grid, mainly for $d \in \{1,2,3\}$, and a local rule determines the state of each location as a function of its own state and the state of its neighbors in the previous time unit. (Indeed, time is also discrete.)

The environment's *evolution in time* is captured by a $d+1$ dimensional array, denoted $\mathtt{ENV} : \mathbb{Z}^{d+1} \to \Sigma$, such that $\mathtt{ENV}_j(i_1,...,i_d) \stackrel{\text{def}}{=} \mathtt{ENV}(j,i_1,..,i_d)$ represents the state of location $(i_1,...,i_d)$ at time j, and $\mathtt{ENV}_j$ is determined by $\mathtt{ENV}_{j-1}$. The set Σ is an arbitrary finite set of possible local states, and the (instantaneous) environment is viewed as an infinite d-dimensional grid. Actually, we shall restrict $\mathtt{ENV}$ to $[t] \times [n_1] \times \cdots \times [n_d]$ or rather to $[t] \times [n]^d$, and postulate that $\mathtt{ENV}$ contains neutral values outside this domain. (The notion of a neutral value is rule-dependent, but at the very least we require that a cell maintain the neutral value as long as all its neighboring cells also hold this value).[1]

An *observer*, who is trying to learn or test the environment, may query its locations at any point in time, but at time $j \in [t]$ it may only obtain values of $\mathtt{ENV}_j : [n]^d \to \Sigma$. That is, the observer is modeled as an oracle machine, but this machine is restricted to make queries that are monotonically non-decreasing with respect to the time value (i.e., the value j in queries of the form $(j,i_1,...,i_d)$). This key feature of the model is captured by the following definition (where $\mathtt{aux}$ represents some auxiliary input that the machine may be given).

Definition 1.1 (time-conforming observers): *An oracle machine T is said to be* time-conforming *if, on input $(t,n,\mathtt{aux})$ and oracle access to $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$, it never makes a query $(j,i_1,...,i_d)$ after making a query $(j',i'_1,...,i'_d)$ such that $j < j'$.*

[1] Jumping ahead, in order to exemplify this notion, we mention that zero is a neutral value for linear rules (as in Theorem 1.9), whereas an empty cell is a neutral value for rules that describe moving objects (as in Theorem 1.10).

In particular, any nonadaptive oracle machine is time-conforming, because its queries can be determined beforehand and made at the appropriate order (i.e., time-wise). This does not mean that time-conforming machines are necessarily nonadaptive (see Theorem 1.7 and Section 4.2).

In general, when the observer queries location $(i_1, ..., i_d) \in [n]^d$ at time $j \in [t]$, it will retrieve the visible part of the state $\mathtt{ENV}_j(i_1, ..., i_d)$, rather than the entire state. That is, the model includes an auxiliary function $V : \Sigma \to \Sigma'$, called a viewing function, such that $V(\sigma)$ is the visible part of the state σ; hence, the query $(j, i_1, ..., i_d)$ is answered by $V(\mathtt{ENV}_j(i_1, ..., i_d))$. We may say that the part of σ not revealed by $V(\sigma)$ is the hidden part of σ. Without loss of generality, we may assume that $\Sigma = Q \times \Sigma'$ and $V(q, \sigma') = \sigma'$ for every $(q, \sigma') \in Q \times \Sigma'$. (In this case q is the hidden part of (q, σ').) We also consider the special case in which the states are fully visible, which is captured by the case that $|Q| = 1$. (In this case, we prefer to view V as an identity function.)

The evolution of the environment is *local* in the sense that the value of $\mathtt{ENV}_j(i_1, ..., i_d)$ is determined by the value of $\mathtt{ENV}_{j-1}$ in positions $\{(i_1 + s_1, ..., i_d + s_d) : s_1, ..., s_d \in \{-1, 0, 1\}\}$. The rule of determining the value, denoted $\Gamma : \Sigma^{3^d} \to \Sigma$, is known.[2] Thus, $\mathtt{ENV}_j(i_1, ..., i_d)$ equals $\Gamma(z_{-1,...,-1}, .., z_{1,...,1})$, where $z_{s_1,...,s_d} = \mathtt{ENV}_{j-1}(i_1 + s_1, ..., i_d + s_d)$ and the sequence of all $(s_1, ..., s_d) \in \{-1, 0, 1\}^d$ appears in some canonical order (e.g., lexicographic order, with $(0, ..., 0)$ in the middle). Indeed, *we model the evolution of the environment as a computation of a d-dimensional cellular automaton.*

The computational problems. The computational problems that we consider are: (1) testing whether the observed evolution of the environment is actually consistent with a fixed known rule, and (2) learning the entire evolution of the environment (i.e., recovering the (visible part of the) states corresponding to all locations at all times). We refer to the standard notions of property testing (cf. [11, 22, 21] as well as Section 2) and PAC learning (cf. [23, 18]), when applied with respect to the uniform distribution on the domain (of the functions in question). The symbol ϵ always denotes the relevant proximity parameter. Since the evolution is determined by the local states at the initial time (i.e., by $\mathtt{ENV}_1$), *testing* is equivalent to asking whether the evolution is consistent with the known rule and some initial global state (i.e., $\mathtt{ENV}_1$), whereas *proper learning*[3] calls for recovering the initial global state.

Definition 1.2 (testing consistency with Γ as viewed via V): *We say that an oracle machine T* tests the consistency of evolving environments with respect to $\Gamma : \Sigma^{3^d} \to \Sigma$ and $V : \Sigma \to \Sigma'$ *if for every $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$ the following holds:*

1. *If $\mathtt{ENV}$ evolves from $\mathtt{ENV}_1$ according to Γ, then $\Pr[T^{V \circ \mathtt{ENV}}(t, n, \epsilon) = 1] \geq 2/3$, where $T^F(\overline{p})$ denotes the execution of T on parameters $\overline{p} = (t, n, \epsilon)$ when given access to the oracle F and $V \circ \mathtt{ENV}$ denotes the composition of V and $\mathtt{ENV}$* (i.e., $(V \circ \mathtt{ENV})(j, i_1, ..., i_d) = V(\mathtt{ENV}(j, i_1, ..., i_d))$).

2. *If $V \circ \mathtt{ENV}$ is ϵ-far from $V \circ \mathtt{ENV}'$ for any environment $\mathtt{ENV}'$ that evolves from the corresponding $\mathtt{ENV}'_1$ according to Γ, then $\Pr[T^{V \circ \mathtt{ENV}}(t, n, \epsilon) = 1] \leq 1/3$, where the distance between $F : [t] \times [n]^d \to \Sigma'$ and $F' : [t] \times [n]^d \to \Sigma'$ equals the fraction of entries on which F and F' disagree* (i.e., $|\{(j, i_1, ..., i_d) \in [t] \times [n]^d : F(j, i_1, ..., i_d) \neq F'(j, i_1, ..., i_d)\}|/tn^d$).

[2] The number of possible rules is a function of Σ and d, which is independent of the size of the environment (i.e., n^d). Therefore, it is possible to run the test for each of the rules separately, without significantly increasing the complexity.

[3] In general, proper learning a concept class requires obtaining a description that has the same format as functions in the concept class. Indeed, here $\mathtt{ENV}_1$ serves as such a description.

If Condition 1 holds with probability 1, then we say that T has one-sided error probability*. If all states are fully visible* (i.e., V is the identity function)*, we may neglect to mention V altogether.*

Note that, on top of oracle access to $V \circ \mathtt{ENV} : [t] \times [n]^d \to \Sigma'$, the tester gets the (duration and size) parameters t, n and the proximity parameter ϵ as explicit inputs. The same applies to learners as defined next.

Definition 1.3 (learning evolution according to Γ via V): *We say that an oracle machine* learns the environment evolving according to $\Gamma : \Sigma^{3^d} \to \Sigma$ and viewed via $V : \Sigma \to \Sigma'$ *if the following holds: On input (t, n, ϵ) and oracle access to $V \circ \mathtt{ENV}$ such that $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$ evolves according to Γ, the oracle machine outputs a function $F : [t] \times [n]^d \to \Sigma'$ that is ϵ-close to $V \circ \mathtt{ENV}$. The learner is said to be* proper *if it outputs $\mathtt{ENV}'$ such that $\mathtt{ENV}' : [t] \times [n]^d \to \Sigma$ is an environment that evolves according to Γ and $V \circ \mathtt{ENV}'$ is ϵ-close to $V \circ \mathtt{ENV}$. Equivalently, we may require the proper learner to output (only) the corresponding $\mathtt{ENV}'_1$.*

We seek *time-conforming* oracle machines that solve the corresponding tasks (of testing and learning). Furthermore, we seek testers and learners that solve the corresponding tasks in *sublinear query complexity*, which we interpret as *making $o(n^d)$ queries at each particular time*. In other words, we seek machines of sublinear *temporal* query complexity.

Definition 1.4 (temporal query complexity): *The* temporal query complexity *of an oracle machine querying $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$ is the maximal number of queries that the machine makes to each $\mathtt{ENV}_j$ ($\forall j \in [t]$).*

Definitions 1.1 and 1.4 capture the time-dependent nature of our goals, and distinguish the current testing and learning problems from the standard testing and learning problems regarding various structures (including those related to $(d+1)$-dimensional arrays). First, whenever the oracle machine does not query the entire oracle $\mathtt{ENV}$, the time-conforming condition restricts its access pattern (i.e., the order in which the machine probes the various entries). Needless to say, this requirement reflects the reality of actual observers of natural phenomena, who are certainly time-conforming, since they cannot inspect the past. Second, the temporal query complexity refers to the number of queries made at each time unit (as compared to n^d), rather than the total number of queries (as compared to $t \cdot n^d$). This requirement reflects the reality of actual observers, who are not only time-conforming, but are also restricted in the number of inspections they can perform at any time unit.

A natural question is whether the time-conforming requirement actually restricts the power of testers. In Section 4.1 (see also Theorem 1.6) we show that this is indeed the case: We demonstrate that the time-conforming requirement makes testing of evolving d-dimensional environments fundamentally different from testing properties of the corresponding $(d+1)$-dimensional array. Specifically, there exist evolution rules Γ for which the time-conforming requirement causes a exponential increase in the query complexity of testers (i.e., an increase from $O(\log n)$ to $n^{\Omega(1)}$). Furthermore, within the context of the time-conforming requirement, restricting the temporal complexity may increase the total complexity (see Theorem 1.8).

Recall that *proper learning implies testing* (cf. [11, Sec. 3.1]), and note that the argument extends to our setting (i.e., when referring to time-conforming machines and their temporal query complexity). Specifically, recall that the argument in [11, Sec. 3.1] suggests that the tester first

learns a hypothesis, and then checks the validity of the hypothesis by an auxiliary sample (which is uniformly distributed in the function's domain). Observe that this auxiliary sample can be chosen a priori, and the adequate queries can be made in due time (even before the learning stage is completed). Hence, one immediately gets –

Theorem 1.5 (proper learning implies testing): *If the environment evolving according to* $\Gamma : \Sigma^{3^d} \to \Sigma$ *as viewed via* $V : \Sigma \to \Sigma'$ *can be properly learned within query complexity* Q (resp., temporal query complexity q)*, then the consistency of environments that evolve with* Γ*, as viewed via* V*, can be tested within query complexity* $Q + O(1/\epsilon)$ (resp., temporal query complexity $q + O\left(\left\lceil \frac{1}{\epsilon \cdot t} \right\rceil\right)$)*. The same holds with respect to the total computational complexity.*[4]

Thus, we present positive results regarding testing only when they improve over the best possible learning results (which typically require a total number of $\Omega(n^d)$ queries). We note that there exist evolution rules for which testing requires as many queries as learning (up to a constant factor), and this holds even if the states are fully visible (see Theorem 1.11).

1.2 A taste of our results

In this section we provide an overview of the results presented in this work. The stated results are followed by rather laconic comments regarding their proofs. More substantial overviews of these proofs are provided in Section 1.3, whereas precise statements, together with complete proofs, are to be found in the technical sections themselves.

Separation results. We start by presenting the two separation results that were mentioned in Section 1.1. Both results are established by using one-dimensional environments (i.e., $d = 1$) and hold with respect to fully visible states (i.e., V is the identity function). Recall that, throughout this paper, the evolving environments are represented as functions from $[t] \times [n]^d$ to Σ. For simplicity, we assume in this section that $t = \Theta(n)$.

The first result establishes the non-triviality of the notion of time-conforming observers by showing that the time-conforming requirement may cause an exponential increase in the query complexity of testers (i.e., an increase from $O(\log n)$ to $n^{\Omega(1)}$).

Theorem 1.6 (on the time-conforming requirement, see Theorem 4.1 for a precise statement): *There exist a constant* $c > 0$ *and an evolution rule* $\Gamma : \Sigma^3 \to \Sigma$ *such that: (1) any* time-conforming *tester of evolution according to* Γ *requires* $\Omega(n^c)$ *queries, but (2) there exists a* (non-time-conforming) *tester of query complexity* $O(\log n) + \mathrm{poly}(1/\epsilon)$ *for this property.*

The evolution rule underlying the proof of Theorem 1.6 designates two n^c-bit long parts of the initial configuration, denoted x and y, and proceeds in two stages. In the first stage, the evolution "commits" to x, by encoding it via an error correcting code. In the second stage, this commitment is deleted, and the evolution commits to y as well as to a single bit of x that is determined by y (i.e., x_i where i is determined by y). In addition, the evolution computes and displays a "locally checkable" proof that ascertains the validity of this bit (with respect to the aforementioned encodings). Hence, valid evolutions correspond to ones in which the commitment that appears in the second stage is

[4] This presumes that the proper learner outputs the entire environment (i.e., $\mathtt{ENV}'$), rather than only its initial configuration (i.e., $\mathtt{ENV}'_1$). Otherwise, we can obtain the entire E' from $\mathtt{ENV}'_1$ in extra time $O(tn)$.

consistent with the commitment that appeared in the first stage. This validity can be tested by a general tester (that is not time-conforming), but not by a time-conforming tester who only learns this information (i.e., i as well as x_i) in the second stage, whereas at this time the commitment to x is not available for inspection.

The second separation result asserts that adaptivity is useful also in the context of time-conforming testers.

Theorem 1.7 (on the benefits of adaptivity, see Theorem 4.2 for a precise statement): *There exist a constant $c > 0$ and an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that: (1) any* nonadaptive *tester of evolution according to Γ requires $\Omega(n^c)$ queries, but (2) there exists* an adaptive (time-conforming) *tester of query complexity $O(\epsilon^{-1} \cdot \log n)$ for this property.*

The proof of Theorem 1.7 is based on the observation that some separations between adaptive and nonadaptive testers that hold in the standard testing model can be translated to analogous results regarding testing evolving environments. Our translation requires the existence of an efficient algorithm for (exactly) deciding satisfaction of the property that is used in the separation result (of the standard model). Furthermore, this decision procedure should be implementable in a way that does not assist testing the property (i.e., testing the evolution of this computation is not more advantageous than testing the object itself).

We also show that restricting the temporal query complexity may cause an increase in the total query complexity. This is only shown with respect to a partial viewing function, leaving the case of a fully viewing function open.

Theorem 1.8 (on the effect of temporal complexity, see Theorem 4.3 for a precise statement): *There exist a constant $c > 0$ and a viewing function $V : \Sigma \to \Sigma'$ such that for any $q(n) < n^c$ there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ for which the following holds: (1) any tester of temporal query complexity $q(n)$ for evolution according to Γ via V requires more than n^c queries, but (2) there exists a* (time-conforming) *tester of query complexity $O(\log n)^2 \cdot q(n) + O(1/\epsilon)$ for this property.*

The proof of Theorem 1.8 uses an evolution rule that generates an instance of a property that is hard to test but easy to verify when given a short proof (akin to the MAP model of Gur and Rothblum [15]). The instance is generated "under the cover" of the hidden part of the state, and the short proof is presented for very few units of time under a very redundant secret sharing format. Hence, this proof can be read only by a tester of sufficiently high temporal query complexity.

Testing evolution according to specific evolution rules. Turning to the study of testing specific evolving environments, we first note that, in general (i.e., for arbitrary evolution rules $\Gamma : \Sigma^3 \to \Sigma$, even for $d = 1$), testing may require as many queries as learning (cf. Theorem 1.11) and that the computational complexity of testing may be NP-Hard (cf. Theorem 3.1). We thus focus our attention on *special classes* of evolution rules. In two natural cases, we obtain testers of lower query complexity than the corresponding learners. Furthermore, these testers are efficient (i.e., their computational complexity is closely related to their query complexity). The first class of evolution rules that we consider is the class of linear rules.

Theorem 1.9 (sublinear time complexity for testing linear rules): *For any $d \geq 1$ and any field Σ of prime order, there exists a constant $\gamma < d$ such that the following holds. For any linear rule*

$\Gamma : \Sigma^{3^d} \to \Sigma$ *there exists a time-conforming oracle machine of* (total) *time complexity* $\mathrm{poly}(\epsilon^{-1}) \cdot n^\gamma$ *that tests the consistency of an evolving environment with respect to* $\Gamma : \Sigma^{3^d} \to \Sigma$ (and the identity viewing function). *Furthermore, the tester is nonadaptive and has one-sided error probability.*

The proof of Theorem 1.9 appears in Section 5. It is based on proving that, on the average, the value of a random location in the evolving environment (i.e., $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$) depends only on $O(n^\gamma)$ locations in the initial configuration. We note that our upper bound on the query (and time) complexity is only mildly lower than $\mathrm{poly}(1/\epsilon) \cdot n^d$ (e.g., for $d = 1$ and $|\Sigma| = 2$ we obtain the bound $O(n^{0.8}/\epsilon)$), and we wonder whether $\mathrm{poly}(\epsilon^{-1} \log n)$ complexity is possible (for all linear rules).

The second class of evolution rules is aimed at capturing the movement of objects in a d-dimensional grid. The following result refers to the case of $d = 1$ and to objects that move in a fixed-speed but stop whenever a collision occurs (i.e., an object continues moving, one cell at a time, until it collides with another object, and when this happens the object stops).

Theorem 1.10 (testing interruptible moving objects, very loosely stated): *Let* $\Gamma : \Sigma^3 \to \Sigma$ *be a local rule that captures the fixed-speed movement of objects in one dimension such that colliding objects stop forever. Then, there exists a time-conforming oracle machine of* (total) *time complexity* $\mathrm{poly}(1/\epsilon)$ *that tests the consistency of evolving environments with respect to* $\Gamma : \Sigma^3 \to \Sigma$ *and the identity viewing function.*

The proof of Theorem 1.10 appears in Section 6.1.1. The corresponding tester makes quite a few checks, which include checking that individual objects move in a fixed speed as long as they don't stop, checking that these objects do not cross each other, and checking global statistics regarding the number of moving and stopping objects within some intervals at some times. The non-triviality of this testing task is reflected in the fact that the tester has *two-sided error probability, and that this is unavoidable for testers of query complexity that is independent of* n. The latter assertion is a corollary of Theorem 6.8, which asserts that *any nonadaptive tester of one-sided error probability for this task must have query complexity* $\Omega(\sqrt{n})$, which in turn implies an $\Omega(\log n)$ lower bound for general testers (with one-sided error probability). We note that the tester used in the proof of Theorem 1.10 is actually nonadaptive.

Hardness results. In contrast to (the rules considered in) Theorems 1.9 and 1.10, there are evolution rules for which testing is not easier than learning. This relative-hardness result holds even when the states are fully visible, which indicates that this restriction (i.e., fully visible states) does not suffice for making testing easier than learning.

Theorem 1.11 (testing may have the same query complexity as learning, see Theorem 3.5 for a precise statement): *There exist a constant* $c > 0$ *and an evolution rule* $\Gamma : \Sigma^3 \to \Sigma$ *such that both testing evolution according to* Γ *and* (proper) *learning evolution according to* Γ *have* (total) *query complexity* $\Theta(n^c)$*, where in both cases we refer to fully visible states and to all sufficiently small constant values of* $\epsilon > 0$.

Theorem 1.11 is proved in Section 3.3. As in the proof of Theorem 1.7, the main observation is that results that hold in the standard model (in this case relations between the complexity of testing and learning) can be translated to analogous results regarding testing evolving environments. Again,

we pick a property for which probing the process of the construction of the object (having the property) does not reveal more than probing the object itself.

Staying within the realm of fully visible states, the following result asserts that, even in this case, the computational complexity of testing may be NP-Hard (with respect to the size of the environment), provided that the temporal query complexity is "significantly sublinear" (where $f(m)$ is significantly sublinear if $f(m) < m^{1-\Omega(1)}$).

Theorem 1.12 (on the computational complexity of testing with sublinear temporal query complexity, see Theorem 3.2 for a precise statement): *Assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, for every constant $c > 0$ there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that no time-conforming probabilistic polynomial-time machine of temporal query complexity n^{1-c} can test whether n-sized environments evolve according to Γ.*

Indeed, Theorem 1.12 stands in contrast to Theorems 1.9 and 1.10, which refer to specific evolution rules. We mention that in the case of a viewing function that hides part of the state, NP-Hardness of testing holds regardless of the query complexity; see Theorem 3.1.

1.3 More on the ideas underlying our proofs

In this section we attempt to give a flavor of the ideas underlying the proofs of our results, going beyond the laconic comments provided in the previous subsection.

1.3.1 On the proofs of our separation and hardness results

Our separation and hardness results (i.e., Theorems 1.6–1.8 and 1.11–1.12, resp.) are all based on the fact that (1-dimensional) cellular automata can emulate (1-tape) Turing machines (while preserving polynomial-time complexity), and thus can generate objects that are hard to test in some sense. The proofs differ by issues such as which computation is emulated and under which circumstances it is emulated.

The basic approach. As a warm-up, let us consider the proof of Theorem 3.1, which asserts that testing the consistency of an evolution with respect to a fixed rule $\Gamma : \Sigma^3 \to \Sigma$ (and a fixed viewing function $V : \Sigma \to \Sigma'$) may be NP-hard, regardless of the query complexity. The basic idea is to design a rule Γ such that evolutions that are consistent with Γ reveal (via the viewing function) an error-corrected encoding of a string if and only if the string is in the NP-set. Hence, membership in the NP-set S can be decided by invoking the tester, while answering its queries (and relying on the fact that strings that are not in S have encodings that are far from the encoding of any string in S). Specifically, we let Γ emulate the verification procedure associated with an NP-witness relation of S, and output (an encoding of) the main input (but not the NP-witness) if the procedure accepts (and output an empty string otherwise). The emulation is carried out using the hidden part of the states, and the output is revealed through the visible part of the states. (The output is maintained for a sufficient number of steps so that evolving environments that output encodings of different strings are far apart (i.e., at constant relative distance of one another).)

As stated upfront, we show that testing evolution under this rule is infeasible by presenting a decision procedure for S, which uses such a hypothetical tester as a black box. Specifically, on input x, the decision procedure for S invokes the tester, while answering queries that correspond

to the emulated computation of the verification procedure by an empty symbol (which matches the expected output of the viewing function), and answers the other queries by using bits in the encoding of x. The key observation is that an evolving environment that "outputs" an encoding of x is legal if and only if there exists an NP-witness w for x (i.e., iff $x \in S$). The distance between evolving environments that output encodings of different strings guarantees that evolving environments that output a string not in S are far from any legal evolution.

The simple argument outlined above relies in an essential manner on the use of a viewing function that hides information. This function reveals the encoding of x, but it does not reveal the emulation of the computation that maps (x, w) to this encoding. Moreover, this simple argument does not allow to separate different query models as asserted in Theorems 1.6 and 1.7. Still, the underlying principle of generating an object that is hard to test in some sense will be pivotal to all subsequent proofs. However, the generation process in these proofs will be less straightforward than in the above case.

Handling the case of fully visible states. Note that Theorem 3.1 cannot possibly hold when the stats are fully visible, since in this case testing consistency of evolutions with a fixed rule is trivial when there are no restrictions on the query complexity. Indeed, Theorem 1.12 refers to (time-conforming) testers of limited query complexity (i.e., sublinear temporal query complexity), and its proof capitalizes on this limitation. In the corresponding cellular automaton, many emulations of the above type take place in parallel, but the temporal query complexity imposed on the tester does not allow it to probe the "informative time" (i.e., the time when the NP-witness is visible) of almost all of these emulations. Specifically, the evolution rule Γ partitions $[n]$ into many blocks (i.e., more blocks than the temporal query complexity), emulates the verification of an NP-witness in each block, identifies the first non-empty output obtained in some block (if such exists), and copies its contents (which is a codeword) to all other blocks. (Otherwise, the evolution maintains an empty output in all blocks.)

We consider evolutions in which a single informative emulation (i.e., an emulation of the verification of a real NP-witness) takes place in one block, selected at random, while dummy emulations (which emulate verification with dummy values) take place in all other blocks. The point is that a time-conforming tester is unlikely to hit the informative block at the informative (emulation) time, whereas if the informative block produces an encoding of a string, then this encoding will be propagated to all other blocks. Hence, the tester should determine whether an evolution that repeats an encoding of x is legal or not, which means that this tester decides membership of x in an NP-set, whereas the tester is highly unlikely to probe the process in which the encoding of x was generated by verifying a valid NP-witness.

Specifically, on input x, the decision procedure invokes the tester, while answering queries that correspond to the emulated computations (in the various blocks) according to the verification of a dummy NP-witness for x (which indeed leads the verification procedure to reject in each block), and answers the other queries by using bits in the encoding of x (which is being copied from a random block that was not probed by the tester during the verification stage).[5] That is, the decision procedure emulates a seemingly illegal evolution; yet, if x is in the NP-set (and the tester did not probe the informative block at the informative time), then the tester's view is consistent

[5] Actually, the decision procedure selects the informative block at random, and aborts the emulation if the tester probes this block during the verification stage. Otherwise, it continues the emulation as if only this block has output the encoding of x.

with a legal evolution that outputs (the encoding of) x. On the other hand, if x is not in the set, then an evolution that outputs the encoding of x is far from being legal.

The forgoing arguments relied on the ability of an ordinary procedure (i.e., the decision procedure we obtain for NP-sets) to answer queries to various fictitious evolutions (or to evolutions taken from a restricted set of possible evolutions). That is, the decision procedure partially emulates the execution of hypothetical testers while providing them with oracle access to fictitious (or restricted) evolutions. This strategy is not available to us when we want to distinguish different types of machines that query a real evolving environment (i.e., different types of testers or testers versus learners). In this case, we must foil oracle machines that actually probe a real evolving environment, and not merely foil hypothetical machines by presenting them with fictitious (or restricted) evolutions that we construct. Furthermore, the evolutions according to the same rule should be testable (or learnable) by a different type of machine.

Consider, for example, the assertion of Theorem 1.11 by which testing requires asymptotically as many queries as (proper) learning. Here we have to present a lower bound on the complexity of testing and match it with an upper bound on the complexity of learning. Furthermore, we claim these bounds for the case of fully visible states. The basic idea is to consider a cellular automaton that computes the inner product (mod 2) of two binary vectors, while placing the vectors and the result in an error correcting form (so as to ensure that there is a sufficiently large distance between different input-output pairs). Using the communication complexity method of [3] (see also [10]), one may show that testing such outcomes requires linear query complexity. But, since we deal with fully visible states, we need to establish this lower bound also with respect to testers that may query the process of computing the outcome.

We first note that the process of encoding the individual vectors poses no additional difficulties, because probing this process only provides values that are functions of one of the vectors (rather than functions of both vectors). Hence, we focus on the process of computing the inner product (mod 2) of the two vectors, and show that this process leaks no additional information when we reduce from the communication complexity of (Unique) `Disjointness` (rather than from `Inner Product`). Specifically, we use the fact that if a pair of vectors has no common 1-entry, then all partial inner products are zero (whereas if the pair has a single common 1-entry, then its inner product (mod 2) is 1).

On separation results regarding various types of testers. We now turn to the separation results (i.e., Theorems 1.6 and 1.7). We start with Theorem 1.7, which asserts a separation between nonadaptive testers and (time-conforming) adaptive testers. The idea here is to present a cellular automaton that can generate (and maintain) instances of some property that is easy to test adaptively but hard to test nonadaptively. We need a process that can generate each string that has the property, but never generates a string that does not have the property. We do *not* need this process to generate these strings with uniform distribution (over the property); any distribution that assigns non-zero weight to each string that has the property will do. For example, we can use a process that generates a random 2-regular k-vertex graph that consists of $k/3$ isolated triangles (or rather maps the set of all $(2k \log_2 k)$-bit strings to the set of all such graphs). This property is easy to test in the bounded-degree graph model (i.e., a constant number of queries suffice), but it cannot be tested by a nonadaptive algorithm that makes $o(\sqrt{k})$ queries [19]. The foregoing process should be "non-informative" in the sense that probing it is not more advantageous than probing the outcome. Specifically, we consider a highly oblivious process that checks whether the input

list of $2k$ vertices corresponds to a k-long sequence of incidence lists of such a k-vertex graph, and produces a dummy graph (e.g., an empty graph) otherwise.

Turning to the proof of Theorem 1.6, recall that this result asserts a gap between the power of (adaptive) time-conforming testers and the power of testers that are not time-conforming. The proof of Theorem 1.6 uses an evolution rule that designates two n^c-bit long parts of the initial configuration, denoted x and y, and proceeds in two stages. In the first stage, the evolution "commits" to x, by encoding it via an error correcting code $C : \{0,1\}^{n^c} \to \{0,1\}^{n/4}$ that has constant relative distance. The point is that this encoding of x occupies a constant fraction of the area of the array representing the evolution of the environment, and so x is effectively committed to by such an environment.

In the second stage, the foregoing commitment is deleted, and the evolution commits to y (via the same code C) as well as to a single bit of x that is determined by y (i.e., x_i where i is determined by y). Specifically, i is computed by applying an extractor F for bit-fixing sources to y; for example, we may interpret y as a sequence over $[n^c]$ and let i be its sum modulo n^c. The encoding of y and the value x_i each occupies a constant fraction of the area of the evolution of the environment, and so y as well as x_i are effectively committed to by such an environment. In addition, the evolution rule realizes a computation of a PCP-of-Proximity proof that ascertains the validity of this bit (with respect to the encodings of x and y), but this proof occupies a small portion of the environment.

Hence, valid evolutions correspond to ones in which the commitment that appears in the second stage is consistent with the commitment that appeared in the first stage. This validity can be tested by a general tester (that is not time-conforming), which uses the PCP oracle in order to verify that i is consistent with the encoding of y (i.e., $i = F(C^{-1}(y))$ and that the claimed value of x_i, denoted v, is consistent with the encoding of x (i.e., $v = C^{-1}(x)_i$). In contrast, a time-conforming tester only learns this bit location (i.e., i as well as x_i) at the second stage, whereas at this time the commitment to x is not available for inspection. Note that F is such that reading a large portion of y keeps $F(y)$ totally undetermined, whereas the tester cannot afford to compare all bits of x to the committed value.

Establishing the results for $t = O(n)$. All the foregoing constructions can be implemented when the evolution time (i.e., t) is polynomial in the size of the environment (i.e., n), since all of them are based on evolutions that emulate polynomial-time computations of 1-tape Turing machines. In order to obtain results that refer to the case that $t = O(n)$, we emulate many computations as above in parallel such that each computation takes place on a block of length ℓ, where the emulation time on ℓ-bit inputs is $\Theta(n)$. The details involve performing some simple manipulations in linear time (on a one-dimensional cellular automaton); for example, the n cells of the automaton can be partitioned into ℓ-cell blocks in $O(n)$ time, and an ℓ-bit string can be copied to the neighboring block in $O(\ell)$ steps of such an automaton.

1.3.2 On testers for special cases

We now turn to our positive testing results, which correspond to two classes of evolution rules (and to the case of fully visible states).

Testers for linear rules. The first result (i.e., Theorem 1.9) refers to the class of all linear rules (over a finite field of prime cardinality). The tester operates by picking a random location $i \in [n]^d$ at a random time $j \in [t]$ and comparing the value of $\texttt{ENV}_j(i)$ to a predetermined linear

combination of the values of the initial configuration (i.e., $\mathtt{ENV}_1(\cdot)$). The crucial fact is that this linear combination is typically sparse, which implies that this tester has sublinear query complexity (i.e., it makes $(n^d)^{1-\Omega(1)}$ queries). Specifically, we prove that on the average, over all times $j \in [t]$, the contents of a cell depends on a sublinear number of locations in the initial configuration (i.e., $(n^d)^{1-\eta}$ locations, for some $\eta > 0$). This is proved by first showing, for every $e \in \mathbb{N}$, that each location $i \in [n]^d$ at time $j \in [t]$ depends only on 3^d locations at time $j-p^e$, where p is the cardinality of the field. Furthermore, each of these locations has the form $i + p^e \cdot \delta$, where $\delta \in \{-1,0,1\}^d$ (i.e., in the case $d = 1$, these locations are $i - p^e$, i and $i + p^e$). Using this fact, we upper bound the average over $j' \in [p^e]$ number of locations at times $j - j'$ that influence the fixed location at time j, by employing a careful accounting of all influences.

We comment that using a similar accounting, one can show that on the average, over all times $j \in [t]$, the contents of a cell does depend on $(n^d)^{\eta'}$ locations in the initial configuration, for some $\eta' > 0$, provided that $t = \Omega(n)$ and the linear rule is not degenerate (see Remark 5.5). This means that, except for linear rules that depend on at most one variable, the above tester has query complexity at least $n^{\Omega(1)}$.

Testers for environments of moving objects. The second result (i.e., Theorem 1.10) refers to a rule that describes the interruptible fixed-speed movement of objects in one dimension. The result asserts a two-sided error tester of complexity that does not depend on the size of the environment. This is complemented by a negative result that asserts that such a complexity cannot be achieved in the case of one-sided error testers. Indeed, our tester performs several checks, and one of these checks refers to certain global statistics (and not to the consistency of a partial view with the rule of movement). The core of the analysis boils down to showing that an evolution that passes all checks with high probability must be close to a legal one. This is proved by a sequence of modifications such that each set of modifications relies on a different check, showing that if the environment passes this check (with high probability) then it is close to one that satisfies the corresponding sub-property. Details follow.

We decouple the property of legal evolutions of environments according to the foregoing rule into the following four conditions:[6] (1) the movement and standing behavior of each object is consecutive; (2) each object appears as standing just after it stopped moving; (3) objects do not cross each other's paths; and (4) objects stop due to an object that occupies the neighboring cell (in their direction of movement). Note that Condition (1) only says that each object moves in some time interval $[1, j]$ and/or stands in some time interval $[j', t]$, but it does not say that if an object stopped moving in location i at time j, then it will appear as standing at location i in time $[j+1, t]$. As indicated by the one-sided error lower bound, it is infeasible to check the latter ("matching") condition for individual objects. Instead, the check for Condition (2) is performed "globally"; that is, by comparing the statistics regarding the movement-stopping times and the standing-start times of all objects. Checks that correspond to Conditions (1) and (3) are performed in a rather straightforward manner, and checking Condition (4) relies on a characterization of the intervals that must be filled with standing objects, given a specific pattern of movement and stopping.

It is rather easy to see that a legal evolution passes all checks with high probability, where the small error probability is due to the statistical checking of Condition (2). The difficult part is

[6] Another condition (i.e., "spaced beginning") was omitted in the current (high-level) description. In fact, the evolution rule does not allow moving objects to start at neighboring locations, and this condition is checked too. The reason for this augmentation is discussed in Section 6.

proving that if an evolution passed all checks (with high probability), then it is close to a legal one. This is shown by a sequence of four modifications, which correspond to the above four checks. For example, it is quite easy to see that if the evolution passes the first check (with high probability), then it must be close to one that satisfies Condition (1). Proving an analogous claim for the other three checks (and corresponding conditions) is more complex. In particular, we should make sure that the modifications that we perform in later stages do not violate the conditions established in prior stages. For example, the matching of movement and standing should be performed while maintaining Condition (1) that asserts that each such behavior occurs in a consecutive interval of time.

1.4 This version

The main difference between this version and prior versions (posted on *ECCC*, as TR14-029 (and its revisions)) is that the separation results captured by Theorems 1.6 and 1.7 are currently stated and proved for fully visible states, whereas in prior versions these results were only established for a partial viewing function. Furthermore, in the case of Theorem 1.6, the gap (between time-conforming and general testers) is widened from sub-exponential to exponential. Needless to say, the proofs are quite different from the original ones, although some of the underlying ideas are similar.

In addition, Theorem 1.8 (on the effect of temporal complexity) is new. Lastly, the overall structure and organization of some of the technical material has been changed. In particular, the material presented in the original Sections 2 and 4 was reorganized and currently appears mainly in Section 3.

1.5 Organization

We start with a short preliminary section (Section 2). Sections 3 and 4 present general results regarding testing and learning with respect to a general local rule Γ. Specifically, in Section 3 we present results that refer to the computational complexity of these tasks and to the difference between the query complexity of the two tasks. Section 4 contains proofs of the two separation results stated in Section 1.2; that is, the separation between time-conforming testers and general ones, and the separation between adaptive and non-adaptive (time-conforming) testers.

Going beyond these generic results of Section 3 requires restricting the scope of the study to specific natural classes of evolution rules. The bulk of this work focuses on two such restrictions: In Section 5 we study the class of *linear rules*, and in Section 6 we study rules that represent environments of moving objects. The type of movement studied in Section 6 is very simple, but, unfortunately, the study turns out to be quite complex.

While the model studied in Section 6 is suitable when the number of objects is linear in the size of the environment (i.e., the "dense case"), a different model is more adequate when the number of objects is relatively very small (i.e., the "sparse case"). The latter model is studied in Section 7. Indeed, in analogy to the study of testing bounded degree graphs (initiated in [12]) as compared to the study of testing dense graphs (initiated in [11]), we need to modify two aspects of the model: the queries allowed (to the testers) and the distance measure (between environments).

Many open problems appear explicitly (or implicitly) in the following sections. The most obvious ones refer to reducing the query complexity of testing evolution according to linear rules

and extending the results regarding moving objects to higher dimensions and to more complex rules of movement. More general directions for further research are mentioned in Section 8.

2 Preliminaries

For the sake of perspective, we recall the standard definition of property testing, which refers to properties of functions. This definition refers to the relative distance between functions, which range over the same domain, which is defined as the fraction of elements on which the functions differ; that is, for $f, g : D \to R$, the relative distance between f and g equals $|\{i \in D : f(i) \neq g(i)\}|/|D|$. We say that f is ϵ-far from the set of functions Π if the relative distance between f and any $g \in \Pi$ is greater than ϵ. Otherwise, we say that f is ϵ-close to Π. The set Π is identified with the property of belonging to it (and the phrase "having the property Π" is often used). For simplicity, we identify the domain of the function with $[s] \stackrel{\text{def}}{=} \{1, ..., s\}$; in general, one may consider arbitrary domains, but in such a case the tester should be provided with a succinct description of the domain.

Definition 2.1 (property testing): *Let $\Pi = \bigcup_{s \in \mathbb{N}} \Pi_s$ be such that Π_s contains functions over the domain $[s]$. We say that an oracle machine T is a tester for Π if for every $s \in \mathbb{N}$ and $\epsilon > 0$ the following holds.*

1. T accepts functions in Π: *For every $f \in \Pi_s$ it holds that $\Pr[T^f(s, \epsilon) = 1] \geq 2/3$, where $T^f(s, \epsilon)$ denotes the execution of T on parameters (s, ϵ) when given access to the oracle f.*
2. T rejects functions that are far from Π: *For every f that is ϵ-far from Π_s, it holds that $\Pr[T^f(s, \epsilon) = 1] \leq 1/3$.*

If Condition 1 holds with probability 1, then we say that T has one-sided error probability.

The query complexity of T is the maximal number of queries that it makes as a function of the input parameters s and ϵ. we say that T is non-adaptive if all its queries are determined as a function of n, ϵ and T's internal coin tosses, obliviously of the oracle's answer to prior queries.

The definition of testing evolving environments (i.e., Definition 1.2) is obtained as a special case of Definition 2.1 by considering functions of the form $\texttt{ENV} : [t] \times [n]^d \to \Sigma$ and properties that correspond to such functions that describe the evolution of an environment according to some local rule Γ, as viewed via V.

Restricting the initial configuration. In may settings it is natural to postulate that not every initial configuration is possible. It is rather natural to postulate that the initial configuration must satisfy some "local conditions" (i.e., conditions that refer to small neighborhoods), and without loss of generality such conditions can be captured by a restriction on the set of possible initial states. Specifically, we restrict the initial configuration to contain only states in a set $\Xi \subset \Sigma$; that is, the initial configuration is a function from $[n^d]$ to Ξ.

The restriction of the initial configuration is instrumental to the hardness and separation results presented in Sections 3.3 and 4, respectively. It is also used in our study of moving objects (presented in Section 6).

Needless to say, when considering environments that evolve from such initial configurations, the testing and learning tasks should be adapted in a natural manner. Here, we spell out the adaptation of Definition 1.2 ("testing"), and the adaptation of Definition 1.3 ("learning") is analogous.

Definition 2.2 (testing evolution from some initial configurations): *For a set of initial states $\Xi \subset \Sigma$, we say that an oracle machine T* tests the consistency of environments that evolve from Ξ^{n^d} according to $\Gamma : \Sigma^{3^d} \to \Sigma$ and viewed via $V : \Sigma \to \Sigma'$ *if for every* $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$ *the following holds:*

1. *If* $\mathtt{ENV}$ *evolves from* $\mathtt{ENV}_1 : [n]^d \to \Xi$ *according to* Γ*, then* $\Pr[T^{V \circ \mathtt{ENV}}(t, n, \epsilon) = 1] \geq 2/3$.
2. *If* $V \circ \mathtt{ENV}$ *is* ϵ*-far from* $V \circ \mathtt{ENV}'$ *for any environment* $\mathtt{ENV}'$ *that evolves from the corresponding* $\mathtt{ENV}'_1 : [n]^d \to \Xi$ *according to* Γ*, then* $\Pr[T^{V \circ \mathtt{ENV}}(t, n, \epsilon) = 1] \leq 1/3$.

In such a case we say that T tests evolution from Ξ^* according to Γ via V.

As stated upfront, the restiction of initial configurations that is captured by Definition 2.2 is expressive enough to enforce any local condition (i.e., any predicate regarding the states of neighboring cells (or cells at constant distance) that must be satisfied in the initial configuration). Furthermore, this convention also enables to "mark" the endpoints of all configurations, by assuming that the part of the initial configuration that is outside of $[n]^d$ contain symbols not in Ξ. Hence, the local rule can identify the non-internal cells (i.e., cells not in $[2, n-1]^d$).

Error correcting codes. Many of the proofs in Sections 3 and 4 utilize error correcting codes and their basic parameters. Here an error correcting code is a mapping $C : \{0,1\}^k \to \{0,1\}^n$, where its length is defined as n and its rate as k/n. The relative distance of the code C is the minimal relative distance between its codewords; that is, it equals the minimum over all distinct $x, y \in \{0,1\}^k$ of $|\{i \in [n] : C(x)_i \neq C(y)_i\}|/n$, where $C(x)_i$ denotes the i^{th} bit of $C(x)$. We say that C, or rather an infinite family of codes, denoted $\{C_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}$, is a good error correcting code if there are consrtants $\delta, \rho > 0$ such that C_k has relative distance at least δ and rate at least ρ.

3 A simple observation and the questions it raises

Recall that we consider a fixed rule, denoted $\Gamma : \Sigma^{3^d} \to \Sigma$, that determines the evolution of the environment such that the value of $\mathtt{ENV}_j(i_1, ..., i_d)$ is determined by applying Γ to the 3^d values in the sequence $\langle \mathtt{ENV}_{j-1}(i_1 + s_1, ..., i_d + s_d) : s_1, ..., s_d \in \{-1, 0, 1\} \rangle$, which is presented in some canonical order. We seek computationally efficient learning algorithms of sublinear temporal query complexity, while noting that one can trivially learn the evolution by querying all n^d entries in $\mathtt{ENV}_1$, and seek even more query-efficient algorithms for the testing task.

The simple observation (alluded to in the section's heading) is that the initial (global) state (i.e., $\mathtt{ENV}_1$) can be *learned* by just considering all $|\Sigma|^{n^d}$ possibilities, and testing the correctness of each possibility by samples; that is, we shall use $O(n^d/\epsilon)$ samples selected uniformly over all times and locations, record the values of these samples, and use them in an exhaustive search that will take place off-line. In other words, we merely use a simple Occam's Razor algorithm, while observing that this algorithm is non-adaptive and spreads its queries (or samples) uniformly among all possible times. This means that we use $O(n^d/\epsilon t)$ queries to each $\mathtt{ENV}_j$, which for $t = \Omega(n)$ yields $O(n^{d-1}/\epsilon)$ queries to each $\mathtt{ENV}_j$.

Note that the foregoing argument did not use the fact that the evolution of the environment is local. Nevertheless, it provides satisfactory results regarding the query complexity of learning,

which cannot be improved (asymptotically) even in the case of local evolution rules of the type we discussed above (except for some degenerate cases). However, the exhaustive search suggested above results in exponential time computations (i.e., exponential in n^d). So one major question is *whether computationally efficient learning algorithms exist in the case of local evolution rules or just in some natural special cases of it*. Another major question is *whether the query complexity of testing may be significantly lower than that of learning, at least in some natural special cases*. We examine both questions next.

The first question is addressed in Sections 3.1 and 3.2. Specifically, in Section 3.1 we show that testing may be computationally hard, and the same holds for proper learning (since the latter implies testing – see Theorem 1.5). In Section 3.2 we show that, in some general cases (e.g., when the states are fully visible), one can obtain more efficient learning algorithms of sublinear temporal query complexity. The same holds for testing (since it is implied by learning). In Section 3.3, we address the second question and show that in some cases the query complexity of testing is not lower than that of learning. The answers provided in these sections refer to the general questions (i.e., they refer to arbitrary local rules). In contrast, Sections 5 and 6 refer to special cases (i.e., specific classes of local rules).

3.1 On the computational hardness of testing

In general, when the states are not fully visible (i.e., the viewing function $V : \Sigma \to \Sigma'$ loses information), efficient testing (let alone efficient learning) is impossible, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$. This is the case since one-dimensional automata can efficiently emulate a one-tape Turing machine. Furthermore, the emulation can be performed via the hidden part of the states while only the final output is visible to the observer (via the visible part of the states).

Theorem 3.1 (on the computational complexity of testing wrt some one-dimensional rules): *There exists a viewing function $V : \Sigma \to \Sigma'$ such that the following holds. For every NP-set S there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that deciding membership* (of $n^{\Omega(1)}$-bit long strings) *in S is probabilistic polynomial-time reducible to testing evolution* (of n-sized environments) *according to Γ via V* (with respect to some constant value of ϵ). *Furthermore, the result holds for any $t = \Omega(n)$.*

Proof: Let R be an NP-witness relation of the set S and let C be an error correcting code with constant relative distance. Consider a one-dimensional automaton, captured by a rule Γ, which emulates a computation of a Turing machine that, on input a pair (x, w), outputs $C(1, x)$ if $(x, w) \in R$ and outputs $C(0, x)$ otherwise. Note that since C is an error correcting code, $C(1, x)$ is far from $C(b, x')$, for every $bx' \in \{0, 1\}^{1+|x|} \setminus \{1x\}$. Here we assume, without loss of generality, that $(x, w) \in R$ implies $|C(1, x)| = |(x, w)|$ and that determining membership in R can be done in linear space (and polynomial-time). Likewise, we assume that C can be computed in polynomial-time and in space linear in its output.

A simple but crucial point is that the viewing function, V, is such that the emulation of the computation is done using the hidden part of the states, whereas the visible part is some fixed ("uninformative") symbol, say $\perp$. On the other hand, once the computation is completed, the output, $C(1, x)$ or $C(0, x)$, is fully visible. Furthermore, the environment repeats the output configuration indefinitely, and so for sufficiently large t, this repeated output dominates the area of the evolving (t-by-n) environment. This essentially "forces the tester to relate" to this (encoded) output rather than to a possible substitution of it.

Hence, if x is not in the NP-set S (charaterized by the witness relation R), then an evolving environment that "shows" $C(1,x)$ (i.e., repeats it in the visible part of the states sufficiently many times) is far from being consistent with the evolution rule Γ. On the other hand, if x is in the set S (characterized by R), then an evolving environment that "shows" $C(1,x)$ is consistent with Γ. Thus, a tester for evolution according to Γ yields a decision procedure for the NP-set S as follows. On input x, the procedure invokes the tester, setting $n = \mathrm{poly}(|x|)$ (and $t = \mathrm{poly}(n)$). It answers queries directed to the "emulation" stage with the adequate "uninformative" symbol (i.e., $\perp$), and answers queries directed to the "repetition" stage of the output with the adequate bit of $C(1,x)$. The procedure accepts x if and only if the tester decides that the environment is consistent with an evolution according to Γ.

The furthermore claim can be proved by considering only initial configurations that are partitioned into blocks (by special symbols), and applying the foregoing cellular automaton separately to each of these blocks. Letting p_1 and p_2 be polynomials such that $(x,w) \in R$ implies $|(x,w)| = p_1(|x|)$ and p_2 upper bounds the emulation time of the foregoing cellular automaton (in terms of $|(x,w)|$), we consider initial configurations that are partitioned into blocks of length ℓ such that $\ell = p_1(k)$ is the largest integer satisfying $p_2(\ell) \leq n$. When wishing to decide membership of $x \in \{0,1\}^k$ in the NP-set S, we emulate a computation that outputs $C(1,x)$ in each of the n/ℓ blocks, which means that $n = \mathrm{poly}(|x|)$. ■

The computational difficulty asserted in Theorem 3.1 is unrelated to the number of queries that may be made to a specific $\mathtt{ENV}_j$. It rather holds regardless of the number of queries made (i.e., it holds also if the entire array $\mathtt{ENV}$ is read), and it arises from the fact that the states are only partially visible. Nevertheless, difficulties arise also in the case that the states are fully visible (i.e., $V(\sigma) = \sigma$ for every $\sigma \in \Sigma$), but in that case they may only arise from the bound on the number of queries to each $\mathtt{ENV}_j$. To illustrate this issue we state the following result, which refers to fully visible states.

Theorem 3.2 (on the computational complexity of testing with sublinear temporal query complexity): *For every NP-set S there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that the following holds. For every constant $c > 0$, deciding membership* (of $n^{\Omega(1)}$-bit long strings) *in S is probabilistic polynomial-time reducible to testing evolution* (of n-sized environments) *according to Γ within temporal query complexity n^{1-c}, where testing is time-conforming and with respect to some constant value of $\epsilon > 0$. We stress that the result holds for fully visible states. Furthermore, the result holds for any $t = \Omega(n)$.*

Theorem 1.12 follows by taking $S \in \mathcal{NP} \setminus \mathcal{BPP}$.

Proof: We start with the construction used in the proof of (the furthermore claim) of Theorem 3.1, which means that we consider initial configurations that are partitioned into blocks (by special symbols), where the cellular automaton maintains this partition. We only consider initial configurations that are partitioned into blocks of length ℓ such that the emulation time on any input of length ℓ, denoted $p_2(\ell)$, is approximately $n^{c/2}$ (i.e., ℓ is the largest integer such that $p_2(\ell) \leq n^{c/2}$). Although the states are fully visible, a tester of temporal query complexity at most n^{1-c} cannot probe most of the ℓ-bit long blocks in any of the first $p_2(\ell)$ time units. This is the case because the total number of blocks probed at any time unit is at most n^{1-c}, whereas the number of blocks is n/ℓ and $p_2(\ell) \cdot n^{1-c} \ll n/\ell$.

The above describes the first stage in the evolution of the the cellular automaton. In the second stage, this cellular automaton checks whether a codeword of the form $C(1,x)$, where $x \in \{0,1\}^k$,

is written in any block, and if so it propagates the first such codewords to all blocks. (This stage can be performed in $O(n)$ steps; see Appendix A.1 for details, and note that determining the first bit encoded in a codeword is easy if we use a systematic code.) In the third stage (which takes most of the time), the cellular automaton just maintains its current configuration (as in the proof of Theorem 3.1).

We decide whether $x \in \{0,1\}^k$ is in the NP-set (defined by R) by invoking the tester and answering its queries as follows, where $\ell = p_1(k)$ and $n = p_2(\ell)$. (We may assume, that $(x, 0^{\ell-|x|}) \notin R$, since otherwise we can immediately decide that x is in the NP-set.) First, we select $i \in [n/\ell]$ uniformly at random. If the tester queries the i^{th} block during the first stage, then we abort, but this happens with small probability (since $p_2(\ell) \cdot n^{1-c} \ll n/\ell$). Otherwise, we answer all queries in the first stage as if $(x, 0^{\ell-|x|})$ is encoded in the initial configuration in each block. We then answer all subsequent queries as if the configuration at the end of the first stage equals $C(0,x)^{i-1}C(1,x)C(0,x)^{(n/\ell)-i}$. In particular, this means that all queries made in the third stage are answered according to $C(1,x)^{n/\ell}$. (By the time-conforming hypothesis, the tester cannot make queries to the first stage after making queries to the second stage, which may reveal i.) When the tester halts, we output its verdict.

Note that if x is in the NP-set, then the answers are according to an evolution that is consistent with the foregoing rule (or automaton); specifically, ignoring the rare event of abort, the answers are according to a legal evolution that starts from an initial configuration in which a valid NP-witness appears (with x) in the i^{th} block and $(x, 0^{\ell-|x|})$ appears in all other blocks, where i is uniformly distributed in $[n/\ell]$. Hence, in this case, the tester must accept with high probability. In contrast, if x is not in the NP-set, then (unless we aborted) the answers are according to an evolution that is far from being legal, since the evolution repeats $C(1,x)^{n/\ell}$ in its third stage (whereas x has no NP-witness). Hence, in this case, the tester must reject with high probability. ■

Remark 3.3 (Theorem 3.2, generalized): *By a straightforward adaptation of the proof of Theorem 3.2, we obtain the following generalization that refers to any polynomial-time computable function $n : \mathbb{N} \to \mathbb{N}$ (e.g., $n(k) = \exp(n^{1/3})$). For every NP-set S there exists a polynomial p and an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ for which the following holds. Deciding membership of k-bit long strings in S is probabilistic $O(n(k)^2)$-time reducible to testing evolution of $n(k)$-sized environments according to Γ within temporal query complexity $n/p(k)$, where testing is time-conforming and with respect to some constant value of $\epsilon > 0$ and $t = \Theta(n)$. Hence, if, for some constant $c > 0$, it holds that $\mathtt{SAT} \notin \text{BPtime}(\exp(k^c))$, where k denotes the instance length, then testing such evolutions in polynomial-time requires temporal query complexity $\Omega(n/(\log n)^{O(1)/c})$.*

The forgoing discussion refers to the general case of a local rule (i.e., Γ) of evolution of d-dimensional environments. However, in special cases (i.e., specific classes of Γ's), there is hope to avoid exponential time computations. Two such special case are that of linear rules and of rules that capture simple types of moving objects (see Sections 5 and 6, respectively).

3.2 On computationally efficient learning

Here we address the question of whether the straightforward learning algorithm (presented in the very beginning of Section 3) can be improved in terms of computational complexity. That is, we seek an algorithm that has sublinear temporal query complexity but avoids the exhaustive search (of possible values of $\mathtt{ENV}_1$) that takes place in this learning algorithm. Theorem 3.2 poses a

limit to what we can hope for (e.g., it is unlikely to obtain $n^{d-\Omega(1)}$ temporal query complexity in polynomial-time), but it does not rule out *any improvement*.

Focusing on the case of fully visible states, observe that we can avoid the exhaustive search by obtaining all values of $\texttt{ENV}_1$, but this violates the requirement that the temporal query complexity be sublinear (i.e., that only $o(n^d)$ queries are made to each $\texttt{ENV}_j$). So the question is whether we can avoid the (full) exhaustive search without making $\Omega(n^d)$ queries to $\texttt{ENV}_1$. The answer is yes. In fact, we present a trade-off between the temporal query complexity (i.e., the number of queries made to each $\texttt{ENV}_j$) and the computational complexity.

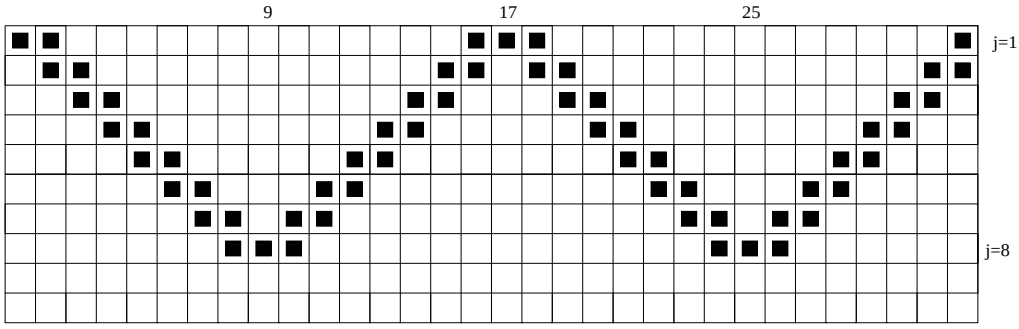


Figure 1: *The saw for $k = 16$ and $n = 32$. (Only 10 time units are shown.)*

It is instructive to start by considering the one-dimensional case. Let k be a free parameter (which governs the trade-off), where k is an even integer. In this case, rather than querying all values of $\texttt{ENV}_1$, we query $\texttt{ENV}$ at a "saw" (see Figure 1); that is, for every i of the form $(i' - 0.5) \cdot k + 1$, where $i' \in [n/k]$, every $j \in [k/2]$ and every $\sigma, \tau \in \{0,1\}$, we query $\texttt{ENV}$ at the point $(j, i + (-1)^\tau \cdot (k/2 - j + \sigma))$. The number of queries made to each $\texttt{ENV}_j$ is at most $4n/k$. This allows us to efficiently derive the value of $\texttt{ENV}$ at any point that comes after this "saw" (in time), and in particular on all points (i,j) for $j \geq k/2$. To obtain the values of the points that precede the saw (in time), we just perform an exhaustive search on each block of length k, but this is an exhaustive search on $|\Sigma^k|$ values (rather than on $|\Sigma^n|$ values).[7] Hence, this learning algorithm offers a trade-off between the number of queries made to each $\texttt{ENV}_j$ and the time complexity.

The "saw"-construction generalizes to any number of dimensions. Specifically, in the case of three dimensions, let $C_\delta(i_1, i_2, i_3)$ denote the set of all grid points that are at max-norm distance at most δ from $(i_1, i_2, i_3) \in [n]^3$; that is,

$$C_\delta(i_1, i_2, i_3) = \{(p_1, p_2, p_3) \in [n]^3 : |p_\ell - i_\ell| \leq \delta \ (\forall \ell)\}.$$

Then, for every $i_1, i_2, i_3 \in \{(i' - 0.5) \cdot k + 1 : i' \in [n/k]\}$ and every $j = 1, ..., k/2$, we query $\texttt{ENV}_j$ at all points in $C_{k/2-j+1}(i_1, i_2, i_3) \setminus C_{k/2-j-1}(i_1, i_2, i_3)$. (In the two-dimensional case, these points can be depicted as the exterior walls of a pyramid, of base length k and height $k/2$.) Hence, we make $(n/k)^3 \cdot ((k - 2j + 3)^3 - (k - 2j - 1)^3) = O(n^3 \cdot (k - 2j + 1)^2/k^3) = O(n^3/k)$ queries to $\texttt{ENV}_j$. In this case, we perform exhaustive searches on $|\Sigma^{k^3}|$ values, and so this learning algorithm offers a trade-off between the number of queries made to each $\texttt{ENV}_j$ and the time complexity. This trade-off improves over the performance of the straightforward learning algorithm. Specifically,

[7] Specifically, for each $i' \in [n/k]$, we conduct an exhaustive search for values of $\texttt{ENV}_1((i'-1)k+1), ..., \texttt{ENV}_1((i'-1)k+k)$ that are consistent with the values of $\texttt{ENV}_j$ at the points $((i' - 0.5)k + 1 + (-1)^\tau(j - \sigma))$, for every $j \in [k/2]$ and every $\sigma, \tau \in \{0,1\}$.

using $k = \sqrt[3]{\log n}$, we get a polynomial-time algorithm that makes $O(n^3/k) = o(n^3)$ queries to each $\texttt{ENV}_j$. In general, we get:

Theorem 3.4 (temporal query complexity vs. computational complexity trade-off): *For every evolution rule $\Gamma : \Sigma^{3^d} \to \Sigma$ and every $k : \mathbb{N} \to \mathbb{N}$ such that $k(n) < t(n)$, there exists an* $\exp(k^d + \log n)$*-time algorithm for* (properly) *learning environments that evolve according to Γ, where the algorithm has temporal query complexity $O(n^d/k)$. We stress that the result is for fully visible states.*

In light of Remark 3.3, it is unlikely that Theorem 3.4 can be significantly improved even for $d = 1$. This is the case because, under plausible assumptions (e.g., SAT not having sub-exponential time algorithms), such evolutions cannot be tested in $\exp(k + \log n)$-time within temporal query complexity $n/\text{poly}(k)$.

Non-proper learning. We note that the ideas outlined above suffice for obtaining an efficient non-proper learning algorithm that makes $O(n^{d-1}/\epsilon)$ queries to each $\texttt{ENV}_j$. To see this, set $k = \epsilon n$, and note that for non-proper learning there is no need to obtain the values of ENV that reside "above" the "saw" (i.e., precede it in time). In particular, we can efficiently recover $\texttt{ENV}_j$ for all $j \geq \epsilon n/2$, and use dummy values for the rest of ENV. (In contrast, proper learning requires obtaining $\texttt{ENV}_1$.) Indeed, this non-proper learning algorithm does not seem to imply an efficient tester.

3.3 On testing versus learning

We first observe that for almost all local rules of evolution of d-dimensional environments, Γ, learning the environment require $\Omega(n^d)$ queries (in total). Yet, in some natural cases (i.e., for some natural rules Γ), testing can be done using $o(n^d)$ queries (in total). Two such cases (i.e., linear rules and rules representing moving objects) are studied in Sections 5 and 6. (We stress that the $\Omega(n^d)$ lower bound on learning holds for these classes too.)

But, as we show next, the query complexity of testing the evolution of environments is not always lower than the query complexity of learning these environment. This result is indeed analogous to known results in the standard models of testing and learning (see [11]), but here we establish it in the context of testing and learning the evolution of environments. Specifically, we prove Theorem 1.11, which we first restate as follows.

Theorem 3.5 (the query complexity of testing may be as high as that of learning): *There exist a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ and a set of initial states $\Xi \subset \Sigma$ such that the following holds.*

1. *Testing evolution from Ξ^* according to Γ has* (total) *query complexity $\Omega(n^c)$.*

   (We stress that this holds even for testers that are not time-conforming.)

2. *Proper learning evolution from Ξ^* according to Γ with respect to constant ϵ has* (total) *query complexity $O(n^c)$. Moreover, the learner is non-adaptive and has temporal query complexity 1.*

We stress that the result holds for fully visible states. Furthermore, the result holds for any $t = \Omega(n)$.

Proof: The main observation is that relations that hold in the standard model (in this case, relations between the complexity of testing and learning), can be translated to analogous results regarding testing evolving environments. The basic idea is to design evolution rules that generate and "display" objects that have a hard-to-test property, where "displaying an object" means maintaining it as the contents of the environment during a sufficiently long time period. Each object having the property will be generated and displayed when starting from a suitable initial configuration, but no initial configuration will cause the displaying of an object that does not have the property.

Recall that we wish to carry out this translation in the context of fully visible states. Thus, we pick a property for which probing the process of the construction of the object (having the property) does not reveal more than probing the object itself. Let $C : \{0,1\}^k \to \{0,1\}^{k'}$ be an explicitly constructed and good error correcting code (i.e., $k' = O(k)$ and the code has constant relative distance). The property that we shall use is

$$\Pi \stackrel{\text{def}}{=} \left\{ \left(C(x), C(y), \sigma^{k'} \right) : x, y \in \{0,1\}^k \ \wedge \ \sigma = \sum_{i \in [k]} x_i y_i \bmod 2 \right\} . \tag{1}$$

For starters, we show (see Claim 3.5.1) that Π is hard to test by combining the ("communication complexity") methodology of [3] with the lower bound of [5]. Later (in Claim 3.5.2), when wishing to show that probing the process of constructing this string does not facilitate testing, we shall consider only inputs for which the *integer* sum $\sum_{i \in [k]} x_i y_i$ is either 0 or 1, which corresponds to the `Unique Disjointness` problem and use the lower bound of [16].

Claim 3.5.1 (warm-up): *Testing* Π *requires* $\Omega(k)$ *queries.*

Proof: Using the methodology of [3] (see also [10]),[8] we reduce the randomized communication complexity problem of computing the inner product mod 2 of two k-bit long strings to testing Π. We refers to the communication complexity ain the joint randomness model, and recall that, by [5, Sec. 4.2], this function has has randomized communication complexity $\Omega(k)$.

Consider parties A and B having inputs x and y, respectively. Using the joint randomness, each of the two parties emulates a copy of the tester for Π, while answering its queries as detailed next. Since they use the same randomness, and provide the same answers to queries (as explained next), they obtain the same final output. The parties collaborate as follows in providing answers to the queries of the tester. Query $i \in [k']$ is answered with the i^{th} bit of $C(x)$, which is sent by A (who can easily compute it based on x) to B (so that B can provide the answer as well to its 'copy' of the tester); query $i \in [k'+1, 2k']$ is answered with the $(i-k')^{\text{th}}$ bit of $C(y)$, which is sent by B to A; and query $i \in [2k'+1, 3k']$ is answered by 0 (by each party). The parties output 0 if the tester accepts and 1 otherwise.

Note that if the inner product of x and y is 0 (mod 2), then $(C(x), C(y), 0^{k'}) \in \Pi$ and the tester accepts with probability at least 2/3; but if the inner product of x and y is 1 (mod 2), then $(C(x), C(y), 0^{k'})$ is $\Omega(1)$-far from Π and the tester rejects with probability at least 2/3. ■

We shall consider environments of length n that evolve in four stages, where each stage takes $\Theta(n)$ steps. For $k = n^c$ and $k' = O(k)$ as above, the environment is partitioned into $n/3k'$ regions, each consisting of $3k'$ cells, as depicted in Figure 2.

[8] Indeed, the actual claim is closely related to a special case of [10, Thm. 4.1].

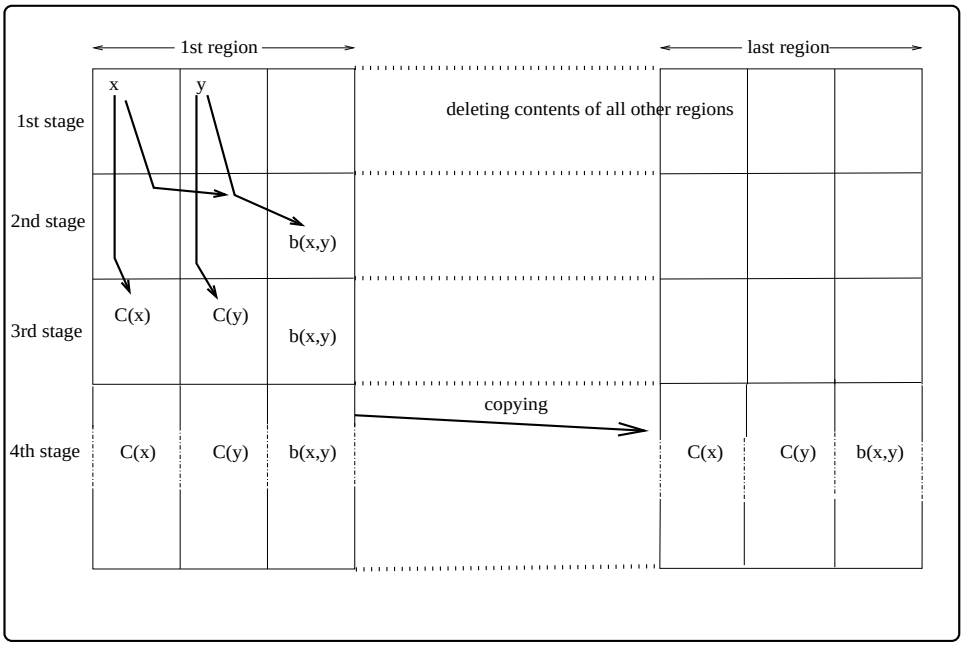


Figure 2: *The evolution of the environment used in the proof of Theorem 3.5. Only the first and last regions are shown; the other regions are identical to the last region.*

- In the first stage, the automaton resets the states of the cells in all regions except the first region (after approximating n and determining $k = n^c$ and $k' = O(k)$ accordingly). (The restriction of the states of the initial configuration to Ξ enables this stage as well as the emulation of Turing machines by a cellular automaton.)

  This resetting is done in order to allow the learning algorithm to ignore the states of the cells in all other regions, and to focus on obtaining the initial values of the remaining $3k'$ cells. (Setting the other initial values to zero yields an evolution that is $O(n^2/nt)$-close to the correct one.)

  We denote the contents encoded by the first k cells by x, and the contents encoded in cells $k'+1, ..., k'+k$ by y.

- In the second stage, the automaton computes $\sigma \leftarrow b(x,y) \stackrel{\text{def}}{=} \sum_{i\in[k]} x_i y_i \bmod 2$, where this computation is performed in k iterations such that in the i^{th} iteration the automaton computes $x_i y_i$ and adds it to the currently accumulating sum $\sum_{j\in[i-1]} x_j y_j \bmod 2$.

  We mention these details, because they mean that an oracle machine that queries the evolving environment at this stage may obtain, in addition to individual bits of x and y (and products $x_j y_j$), only each of these partial sums (i.e., $\sum_{j\in[i]} x_j y_j \bmod 2$) by making one suitable query. Hence, we may assume, without loss of generality, that queries at this stage return such partial sums.

  The second stage is completed by storing the final outcome (i.e., σ) in each of the last k' cells of the first region. The total number of steps in this stage is $O(k^2)$, but this is $O(n)$ by our choice of $c > 0$.

- In the third stage, the automaton computes $u \leftarrow C(x)$ and $w \leftarrow C(y)$, and stores the results in the first $2k'$ cells of the first region.

  This computation takes poly(k) steps, but this is $O(n)$ by our choice of $c > 0$. We note that an oracle machine that queries the evolution of this stage may obtain arbitrary Boolean functions of either x or y, but nothing else (assuming, without loss of generality, that it has already obtained the value of σ and thus does not query for it at the current stage).

- In the last (i.e., fourth) stage, the automaton replicates the contents of the first region (i.e., $(u, w, \sigma^{k'})$) in space and time. That is, the automaton copies the contents of the first region to all other regions, and propagate this contents for the rest of the evolution. Thus, the value $(C(x), C(y), b(x,y)^{k'})$ is replicated $(n/k') \cdot (t - O(n))$ times, and makes up most of the area of the evolving (t-by-n) environment.

It follows that proper learning is possible by performing $2k$ queries. This is done merely by querying the values of the bits of x and y, and relying on the fact that these values determine all but a $O(n)/t$ fraction of the (t-step) evolving environment; that is, the learner may set all other values of the initial environment to zero, yielding an evolution that is $O(n^2/tn)$-close to the actual one. (Note that the effect of this resetting of the values of the initial environment is restricted to the first stage, since any trace of these values is erased during that stage.) Part 1 of Theorem 3.5 follows. (Furthermore, we can obtain temporal query complexity 1, by having this learner read these $2k$ bits (non-adaptively) at different times (within the first stage).)

From this point on we focus on showing that testing this evolution requires $\Omega(k)$ queries. The key observation is that each query made to the evolving environment may yield either a Boolean function of x or a Boolean function of y or the value $\sum_{j \in [i]} x_j y_j \bmod 2$ for some $i \in [k]$. Furthermore, the aforementioned Boolean functions belong to a predetermined set of $m \stackrel{\text{def}}{=} O(kn)$ functions, denoted $f_1, ..., f_m$. (The latter fact is not essential, but it makes the formulation of the next result easier.)[9]

Claim 3.5.2 (a standard testing lower bound): *For every $i \in [k]$, let $b_i(x,y) \stackrel{\text{def}}{=} \sum_{j \in [i]} x_j y_j \bmod 2$. Then, testing Π' requires $\Omega(k)$ queries, where*

$$\Pi' \stackrel{\text{def}}{=} \left\{ \left(F(x), F(y), B(x,y)^{O(k)}, C(x)^{n^2/k'}, C(y)^{n^2/k'}, b(x,y)^{tn/4} \right) : x, y \in \{0,1\}^k \right\} \qquad (2)$$

such that $F(z) = (f_1(z), ..., f_m(z))$ and $B(x,y) = (b_1(x,y), ..., b_k(x,y))$.

Note that the bits of the tested string correspond to the various queries that the evolution tester can make to the evolving environment. The $tn/4$ repeats of $b(x,y)$ represent the third of the area of the last stage (which is occupied by the value $b(x,y)$).

Proof: Using the methodology of [3], we reduce the communication complexity of `Unique Disjointness` to testing Π'. Recall that in `Unique Disjointness` the two parties are given inputs x and y, respectively, such that $I \stackrel{\text{def}}{=} \{i \in [k] : x_i = y_i = 1\}$ has cardinality at most 1. They need to decide whether or not $I = \emptyset$. Note that under the promise that $|I| \leq 1$, the question reduces to computing $b(x,y)$, since $I = \emptyset$ implies $b(x,y) = 0$ whereas $|I| = 1$ implies $b(x,y) = 1$. (Indeed, in general

[9] An alternative presentation may avoid the presentation of an explicit property testing problem, and apply the methodology of [3] without sticking to its specific formulation (as done in [10]).

$|I| = \sum_{i \in [k]} x_i y_i$ (over the integers).) Recall that the communication complexity of `Disjointness` is $\Omega(k)$; cf. [16].

The reduction of the communication complexity of `Unique Disjointness` to testing Π', proceeds as follows, where the parties A and B hold the inputs x and y, respectively. Similarly to what was shown in the proof of Claim 3.5.1, the parties emulate a tester for Π' by answering its queries. Specifically, queries of the form $f_i(x)$ (or $C(x)_i$) are answered by A, queries of the form $f_i(y)$ (or $C(y)_i$) are answered by B, and queries of the form $b_i(x,y)$ are answered by 0 (by both parties). (That is, each party handles queries that are functions of its own input only, and the only allowed queries that refer to both inputs are answered by a predetermined default value.) The parties output 1 (indicating that $I = \emptyset$) if the tester accepts and 0 otherwise.

Note that if $b(x,y) = 0$, then $b_i(x,y) = 0$ for all $i \in [k]$, and it follows that

$$(F(x), F(y), 0^{O(k^2)}, C(x)^{n^2/k'}, C(y)^{n^2/k'}, 0^{tn/4}) \in \Pi' \ .$$

In this case the tester, which was given oracle access to this very input, accepts with probability at least 2/3, and the parties will output 1 (indicating that $I = \emptyset$). On the other hand, if $b(x,y) = 1$, then $(F(x), F(y), 0^{O(k^2)}, C(x)^{n^2/k'}, C(y)^{n^2/k'}, 0^{tn/4})$ is $\Omega(1)$-far from Π', since the distance is dominated by the replicated codewords $C(x)$ and $C(y)$ and the replicated value of zero that appears instead of the replications of $b(x,y) = 1$. In this case the tester, which was given oracle access to this very input, rejects with probability at least 2/3, and the parties will output 0 (indicating that $I \neq \emptyset$). ■

Using the correspondence between testing Π' and testing the evolution according to the foregoing evolution rule Γ, Part 2 of Theorem 3.5 follows.

Turing to the furthermore claim, we note that, by our choice of k, all computations can be emulated in $O(n)$ time by a one-dimensional cellular automaton with n cells. The details (presented in Section A.1) include determining n, and copying a block of m bits from one region of the automaton to another in $O(m)$ steps of the automaton.[10] ■

4 Two separations and one effect

In Section 4.1, we demonstrate that the time-conforming requirement makes testing of evolving d-dimensional environments fundamentally different from testing properties of the corresponding $(d+1)$-dimensional array. In Section 4.2, we demonstrate that adaptivity can significantly reduce the query complexity also in the context of time-conforming testers. Both separations are proved for the case that the states are fully visible. Lastly, in Section 4.3 we demonstrate the effect of restricting the temporal complexity of testers on the total query complexity. This result uses a partial viewing function. All results presented in this section are proved for the case that the initial configuration satisfies some local condition, as postulated in Definition 2.2, which is introduced and discussed in Section 2.

4.1 Time-conforming testers versus general testers

A natural question is whether the time-conforming requirement actually restricts the power of testers. Since any nonadaptive tester is (or can be made) time-conforming, a separation may exist

[10] Actually, we compute $2^{\lfloor \log_2 n \rfloor}$ and use this value rather than n. The basic step here is computing $\log_2 n$ by repeated bisections. Regarding the copying task, a crucial task is to locate the distant endpoint of the adjacent block.

only via adaptive testers.

Theorem 4.1 (on the time-conforming requirement wrt some one-dimensional rules): *There exist a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, and $\Xi \subset \Sigma$ such that the following holds.*

1. *Evolution from Ξ^* according to Γ can be tested using $\mathrm{poly}(\epsilon^{-1}) + O(\log n)$ queries.*
2. *Evolution from Ξ^* according to Γ cannot be tested by a* time-conforming *oracle machine that makes $o(n^c)$ queries.*

We stress that the result holds for fully visible states. The result holds for any $t = \Omega(n)$.

Needless to say, the tester in Item 1 is not time-conforming, which in particular means that it is inherently adaptive. The construction uses PCP-of-Proximity as defined and constructed by Ben-Sasson *et al.* [2], (cf., also, the related assignment testers of [6]). We provide a definition of this notion in the course of the proof.[11]

Proof: Let us first outline the high level structure of our construction of a suitable evolution rule Γ. We shall consider environments of length n, which are partitioned into four $n/4$-cell long regions, and evolutions that proceed in two stages, each having a duration of $\Theta(n)$ time units.

For a constant $c > 0$ to be determined so that various $\mathrm{poly}(n^c)$-time computations take $n^{1/2}$ time, let $k = n^c$. The first region initially holds two k-bit long strings, denoted x and y, and at the second stage it will also contain a PCPP proof oracle (as explained in detail subsequently). The second and third regions will hold encoding of x and y, respectively, and the fourth region will hold an encoding of (i, x_i), where $i \in [k]$ will be computed during the second stage. These encodings are not present at the initial configuration; they will be computed at different times and will be displayed at some time periods, where "displayed" means being maintained as the contents of the environment during that time period.

Specifically, the encoding of x will be created and displayed at the first stage (or "time period"), and will be deleted at the very beginning of the second stage. Only then, will the encoding of y be computed and displayed, and ditto regarding the encoding of (i, x_i), where $i = F(y)$ (for a function F to be determined later) will also be computed in the second stage. We stress that x and y themselves will continue to appear in the first region, but their appearance there is not "robust" (i.e., an environment containing their values is very close to one that contains different values). In contrast, an environment that displays the encoding of x (ditto of y) is far from an environment that displays different values of these strings, because the encoding is via an error correcting code of relative constant distance. In addition, a PCP-of-Proximity proof that (i, x_i) matches the encodings of x and y will be computed in the second stage and displayed for a short time in first region. All this is schematically depicted in Figure 3.

We now turn to a more detailed description of the evolution of environments under the evolution rule Γ being designed here. We shall use an error correction code $C : \{0,1\}^k \to \{0,1\}^{n/4}$ that has constant relative distance and a linear-time encoding algorithm. Furthermore, we shall rely on the fact that a one-dimensional cellular automaton can realize the mapping $z \mapsto C(z)$ in linear time. This can be done by letting $C(z) = C_0(z)^{n/4n'}$, where $C_0 : \{0,1\}^k \to \{0,1\}^{n'}$ has constant relative

[11] Our original proof, which referred to a partial viewing function, relied on results of Gur and Rothblum [15] (as well as on codes that are locally testable and decodable using a polylogarithmic number of queries).

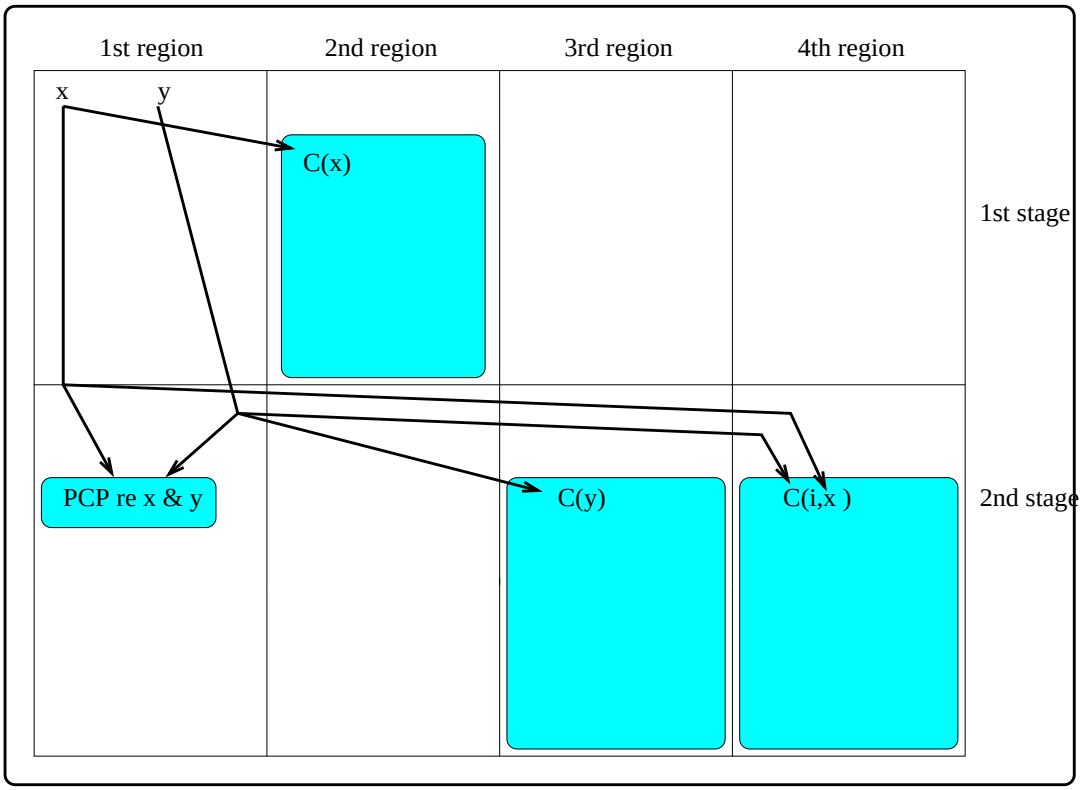


Figure 3: *The evolution of the environment used in the proof of Theorem 4.1, where C denotes the commitment/encoding and $i = F(y)$. The rounded rectangles represent areas that display values, and the areas outside them (with the exception of areas at the vicinity of arrows) are inactive.*

distance and is poly(k)-time computable.[12] In addition, we shall use a polynomial-time computable function $F : \{0,1\}^k \to [k]$, which will also be specified later. (We hint that F will be a function that is hard to predict based on $o(k/\log k)$ bits of its random input.)

In the first stage, the k-bit long string x that resides in (say, the first k cells of) the first region is transmitted to the second region, where it is converted into its encoding $C(x)$ and maintained there for $\Omega(n)$ time units. Hence, an error correcting version of x occupies a constant fraction of the entire t-by-n array that describes the evolution of the environment. Indeed, we need to show that one-dimensional cellular automata can implement this stage in $O(n)$ time. (This would have been much easier if we were allowed time $t = \text{poly}(n)$.)[13] Both in this stage and in the second stage, parts of the environment for which we prescribe no action are supposed to maintain their state; we view such parts as inactive.

In the second stage, the contents of the second region is deleted, and only once this deletion process is completed, the following computations take place. Firstly, the k-bit long string y that resides in (say, cells $k+1$ through $2k$ of) the first region is transmitted to the third region, where it

[12] We use the fact that a one-dimensional cellular automaton can realize the mapping $w \mapsto w^t$ in $O(t \cdot |w|)$ time. See Appendix A.1 for details.

[13] In such a case, each stage would have had a duration of $\Theta(t)$, and contents would have remained revealed for $\Theta(t)$ time units.

is converted into its encoding $C(y)$ and maintained there for $\Omega(n)$ time units. Secondly, the value $i = F(y)$ is computed, and an encoding of (i, x_i) is placed in the fourth region and maintained there for $\Omega(n)$ time units. This encoding uses an error correcting code $C' : \{0,1\}^{\log_2 k+1} \to \{0,1\}^{O(\log k)}$ that has constant relative distance and an exponential-time[14] encoding algorithm, and we encode (i, x_i) by $C'(i, x_i)^t$, where $t = 0.25n/|C'(i, x_i)|$. Lastly, the evolution computes a PCP-of-Proximity (PCPP) proof ascertaining that (i, x_i) is indeed consistent with $C(x)$ and $C(y)$, where $C(x)$ and $C(y)$ are viewed as input oracles (and (i, x_i) may be viewed either as an explicit input or as an input oracle).[15] This proof is computed in the first region and presented there for a short period of time (so that it occupies a negligible portion of the area of the entire array). Again, we need to show that one-dimensional cellular automata can implement this stage in $O(n)$ time.

We next explain why this construction establishes the separation claim made in the theorem. We first show that a general tester (which is not time-conforming) can test whether an evolution is consistent with the foregoing rule Γ by making logarithmically many queries. The reason is that such a tester can first retrieve i and x_i from the fourth region, by querying the suitable locations at a time that correspond to the (displaying part) of the second stage, and then check the consistency of (i, x_i) with $C(x)$ and $C(y)$, by querying the PCPP proof as well as $C(x)$ and $C(y)$. (In addition, this tester also check that $C'(i, x_i)$ is repeated in the relevant part of the fourth region.) We observe that there is no need to check the validity of the various computational processes, since these can be made to occupy a negligible fraction of the entire area of thearray. Also note that the fact that $C(x)$ appears only in the first stage does not hinder the general tester (which is not time-conforming). Hence, the query complexity of this tester is $|C'(i, x_i)| + O(1) = O(\log n)$, where the first term is due to reading (i, x_i) from the fourth region and the second term is due to the PCPP verifier (and to the repetition test).

In contrast, no time-conforming machine can test the consistency of the evolution with the foregoing rule Γ by making $o(k/\log k) = o(n^{1/c})$ queries. Intuitively, this is the case because by the time i and x_i are easy to obtain (i.e., at the second stage), the encoding of x is no longer available. During the first stage, a time-conforming tester can only obtain $o(k)$ bits of information regarding x and $o(k/\log k)$ (actual) bits of y, but no function that combines both x and y is present in the evolution at that stage. Furthermore, we use a function F such that conditioned on any $(k/\log_2 k) - 1$ bits of a random y, the bit $F(y)$ is random (e.g., F views y as a $k/\log_2 k$-long sequence of numbers in $[k]$ and computes their sum modulo k). This guarantees that the value of $F(y)$ is unpredictable based on $o(k/\log k)$ bits of y, when y is uniformly distributed in $\{0,1\}^k$. (Indeed, F is a randomness extractor for bit-fixing sources that is resilient to the fixing of $k-1$ bits [4]).[16] In this case, if we view y and x as being uniformly distributed in $\{0,1\}^k$, then the tester has $o(1)$ bits of information regarding $x_{F(y)}$, and it cannot distinguish the case that $(x_{F(y)}, F(y))$ is presented from the case that $(x_{F(y)}, F(y) \oplus 1)$ is presented. Needless to say, this intuition should be turned into an actual proof. But before getting there, let us be a bit more specific about the various ingredients in use and the implementation of the various computations.

Our requirements from the codes in use are very mild; for example, we only required C and C' to be computable in polynomial-time and exponential time, respectively, although such codes can be

[14] Here we state the minimal requirements from C', while noting that much better codes are known (e.g., obtaining almost-linear-time encoding is easy).

[15] In the latter case, we refer to a definition of PCPP that gives equal weight to the inputs provided by each of its input oracles (which, in this case, are three in number).

[16] We mention that we may actually use an extractor that is resilient to the fixing of any $k-1$ bits; for example, we may use $F(y) = yG^\top$, where G is the $(\log_2 k)$-by-k generator matrix of the Hadamard code (see [4]).

computed in nearly linear time. As a function $F : \{0,1\}^k \to [k]$, we can just use $F(z_1, ..., z_{k/\log_2 k}) = (\sum_{i \in [k/\log_2 k]} z_i \bmod k) + 1$, where $z_1, ..., z_{k/\log_2 k} \in \{0,1\}^{\log_2 k} \equiv [k]$. The less trivial ingredient is a PCP-of-Proximity (PCPP) verifier, which is an oracle machine that verifies a property of the first oracle (i.e., the input oracle) using a proof that is provided in a second oracle, while making a constant number of queries. In our case, the input oracle will be the pair $(C(x), C(y))$, and the proof oracle will be placed in the first region in the beginning of the second stage. In addition, this machine can get an explicit input, which in our case will be the pair (i, x_i). Such an oracle machine, called a verifier, makes $O(1/\epsilon)$ many queries and satisfies the following completeness and soundness conditions.

Completeness: If the input oracle has the property, then there exists a proof oracle that makes the verifier accept with probability 1.

Soundness: If the input oracle is ϵ-far the property, then the verifier rejects with probability at least $1/2$, no matter which false proof is provided as a proof oracle,

Such PCPP systems, with a proof length that is polynomial in the length of the input oracle, are known to exist for any set in $\mathcal{NP}$ (cf., e.g., [2, 6]). Furthermore, for properties in $\mathcal{P}$ (as is the case here), proof oracles as guaranteed in the completeness condition can be constructed (based on the input) in polynomial time. Lastly, we mention that we choose k such that all computations can be implemented by a cellular automaton with n cells in time $O(n)$ (see comment at the end of the proof of Theorem 3.5).

As stated above, a general tester (which is not time-conforming) can test the consistency of the environment with the evolution rule Γ by making $\mathrm{poly}(\epsilon^{-1}) + \log n$ queries. This tester first reads $C'(i, x_i)$ from the fourth region by querying it at a time that correspond to the (displaying part) of the second stage. Next, it checks that the repetition structure of the encoding of $C'(i, x)$ (as well as of $C(x)$ and $C(y)$) is indeed satisfied by the relevant parts. Finally, it invokes the PCPP verifier by providing it with (i, x_i) as explicit input and emulating the input and proof oracles that it tries to access. In particular, the proof oracle is emulated by querying the relevant portion that appears in the second stage (in the first region). Queries to the first part of the input oracle (i.e., the part that corresponds to $C(x)$) are answered by making queries to the second region at random times that correspond to the first stage. Here is where we dramatically violate the time-conforming condition; these queries to the first stage are made after we have retrieved the value of (i, x_i) from the second stage. Lastly, queries to the second part of the input oracle (i.e., the part that corresponds to $C(y)$) are answered by making queries to the third region at random times that correspond to the second stage.

Note that we do not check whether the PCPP proof itself is valid, let alone is the "canonical" one that should have been computed according to the transformation (see discussion in [14]), but rather rely on its occupying a negligible portion of the area of the array (say at most $n^{3/2}$ cells at all times).[17] Likewise, we do not check the validity of the *process* of computing this PCPP proof nor the process of computing the functions F, C and C' (or the process of transporting x and y). What we do check is the displayed values of $C(x)$, $C(y)$ and $C'(i, x_i)$ (as well as areas that are supposed to be "inactive"; see Figure 3).[18] Hence, we get

[17] If the proximity parameter is smaller than this fraction (i.e., $\epsilon < n^{-1/2}$), then we read the entire array, while relying on $O(n^2) = \mathrm{poly}(1/\epsilon)$.

[18] Here we refer to parts of the environment for which we prescribe no action. Recall that these regions are supposed to maintain their state, and the tester should check this too.

Claim 4.1.1 (fast but non-time-conforming testing): *The consistency of the environment with* Γ *can be tested using* $\mathrm{poly}(\epsilon^{-1}) + \log n$ *queries, by a machine that is not time-conforming.*

We next prove that any time-conforming tester for the consistency of the environment with the evolution rule Γ must make $\Omega(n^c)$ queries. Towards this end, we consider two distributions of environments. The first distribution is obtained by selecting uniformly at random $x, y \in \{0,1\}^k$, and considering a proper evolution (according to Γ) that starts from an initial configuration in which x and y appear in the first $2k$ cells. The second distribution is similar, except that after the contents of $C(x)$ is deleted (at the beginning of the second stage), the value of the i^{th} bit of x is flipped.

Now, on the one hand, the first distribution is supported by valid (according to Γ) evolutions, whereas the second distribution is supported by arrays that are $\Omega(1)$-far from representing a valid evolution. The latter assertion is based on the fact that each of the encodings of x, y and $(i, x_i \oplus 1)$ by an error correcting code of relative constant distance, occupies a constant fraction of the area of the array, whereas i is determined by y. On the other hand, we claim that a time-conforming machine of query complexity $o(k/\log k)$ cannot distinguish these two distributions.

Claim 4.1.2 (indistinguishability claim): *Let* M *be a time-conforming machine of query complexity* q*, and denote by* p_i *the probability that it outputs 1 after querying the* i^{th} *distribution (for* $i \in \{1,2\}$*). Then,* $|p_1 - p_2| = O(\sqrt{q/k'})$*, where* $k' = k/\log_2 k$*.*

Proof: We consider the information available to such a machine just after the deletion of $C(x)$ is completed, at the beginning of the second stage, which is the very time in which the i^{th} bit of x is flipped in the second distribution. Assuming that $q < k'$, it follows that conditioned on the information gathered by M, the value of i is still uniformly distributed in $[k]$, since M gathered at most $k' - 1$ bits of y whereas $F(y)$ is uniformly distributed (in $[k]$) even when conditioning on the values of $k' - 1$ bits of y. Note that this assertion relies on the fact that queries made until this time (when x_i is flipped) can only reveal physical bits of y rather than arbitrary information (of bounded amount) on y. This is the case since no computation is performed on y until this time.

The q queries made until this time may reveal at most q bits of information about x, which means that q/k bits of information are revealed on the average regarding a random bit of x. Denoting by ϵ_i the amount of information revealed on x_i (i.e., one minus the entropy of x_i given these q answers), it follows that the probability that we can guess x_i with probability at most $0.5 + O(\sqrt{\epsilon_i})$. Hence, $|p_1 - p_2| = \sum_{i \in [k]} O(\sqrt{\epsilon_i})/k$, and the claim follows (since $\sum_{i \in [k]} \epsilon_i/k \leq q/k$). ■

Having shown (in Claim 4.1.2) that a time-conforming machine of query complexity $o(k/\log k)$ cannot distinguish a distribution of valid evolutions from a distribution of arrays that are $\Omega(1)$-far from representing a valid evolution, we have established Part 2 of Theorem 4.1. This completes the proof of Theorem 4.1 ■

4.2 Adaptive versus nonadaptive testers

Confining ourselves to the context of time-conforming testers, we observe that also in this setting adaptive testers may be much more efficient than nonadaptive ones.

Theorem 4.2 (on the benefits of adaptivity wrt some one-dimensional rules): *There exist a constant* $c > 0$*, an evolution rule* $\Gamma : \Sigma^3 \to \Sigma$*, and* $\Xi \subset \Sigma$ *such that the following holds.*

1. *Evolution from Ξ^* according to Γ can be tested by a time-conforming oracle machine that makes $O(\epsilon^{-1}\log n)$ queries.*

2. *Evolution from Ξ^* according to Γ cannot be tested by a* nonadaptive *oracle machine that makes $o(n^c)$ queries.*

We stress that the result holds for fully visible states. The result holds for any $t = \Omega(n)$.

Proof: The main observation is that separations between adaptive and nonadaptive testers that hold in the standard testing model can be translated to analogous results regarding testing evolving environments. Our translation requires the existence of an efficient algorithm for sampling (objects having) the property that is used in the separation in the standard model. This sampler need not produce the uniform distribution over objects having the property, but the support of its output distribution should equal the set of all objects having this property.

We find it simplest to use a result of [19] that implies that *the nonadaptive query complexity of testing any non-trivial property of d-regular graphs is $\Omega(\sqrt{k})$*, where testing refers to the bounded degree model of [12] and a property is called non-trivial if for any k there exists a k-vertex d-regular graph that has the property and a k-vertex d-regular graph that is $\Omega(1)$-far from having the property.[19]

We shall use the following property, denoted Π_d, which consists of all d-regular graphs that are composed of isolated $(d+1)$-vertex cliques. Any constant $d \geq 2$ will do. Note that Π_d is non-trivial and that it can be tested (in the bounded-degree model) in complexity $O(d^2/\epsilon)$ (by selecting $O(1/\epsilon)$ random vertices and exploring their depth-2 neighborhood). The property Π_d is quite easy to sample; we do so by selecting a random $d \cdot k$-long sequence over $[k]$, and outputting this sequence if and only if it corresponds to a sequence of incidence lists of a k-vertex graph in Π_d. (Otherwise, we output a dummy sequence, which may be thought of as representing an empty graph.)

We shall describe the construction for some $t = \mathrm{poly}(n)$, but it can be adapted to the case of $t = O(n)$ by using ideas as in the proof of Theorem 4.1. The evolution will consist of two stages such that in the first stage the evolution constructs a graph (or rather verifies that the randomness corresponds to a legal representation of a graph in Π_d) and in the second stage this graph is merely displayed (or maintained). The first stage will take $o(t)$ time units (say $t^{1/2}$ time units), whereas the second stage will be far longer and dominate the total amount of time.

The initial configuration of the cellular automaton will encode the randomness used by the foregoing sampler, which means that we view the initial configuration as consisting of k lists, each of length d, of elements in $[k]$. (The i^{th} list is supposed to correspond to the list of neighbors of vertex $i \in [k]$, in a d-regular graph.) The crucial point is to perform the foregoing verification (i.e., check that these lists properly represent a graph in Π_d) in a way that does not facilitate testing that the evolution is valid. Specifically, we proceed in k^2 steps such that in the $(i,j)^{\text{th}}$ step the alleged d-long list L_i of neighbors of vertex i is sent to the j^{th} area (which is supposed to occupy the list of vertex j), and the consistency of these two lists is checked, where the lists L_i and L_j, viewed as multi-sets, are consistent if and only if $L_i \cup \{i\}$ and $L_j \cup \{j\}$ are $(d+1)$-subsets of $[k]$ that are either identical or disjoint. The j^{th} area maintains a bit, called its flag, indicating whether all checks done so far have passed. This bit is initially set to 1 and is reset to 0 once some check fails. At the end of the first stage, the evolution computes the conjunction of these flag bits. If the result

[19] The result in [19] is more general.

is 0, then it replaces all lists by dummy lists, and otherwise it replaces each element in each list by its encoding under a good error-correcting code (of constant rate and constant relative distance).

A word about the format of the output of this process is in place. We wish the output format to be such that it preserves distances with respect to the bounded-degree graph model; that is, if a graph G is ϵ-far from the graph G', then the relative distance between the strings that represent them should be $\Theta(\epsilon)$. In addition, we want each query to a k-vertex graph to be emulated by few (i.e., $O(\log k)$) binary queries to its representation. This is achieved by the foregoing encoding.

The adaptive tester ignores the first stage, and only checks that the second stage displays a bit string that represents a graph in Π_d (under the foregoing format). Specifically, it checks that: (1) a single string is maintained in the second stage, and (2) this string encodes a graph that has property Π_d. Condition (1) is checked by simple sampling, where $O(1/\epsilon)$ samples suffice for this, while Condition (2) is checked by emulating a tester for Π_d. Note that each query of the latter tester is emulated by $O(\log k)$ queries to the string that encodes a k-vertex graph. Hence, testing consistency of the evolving environments with the foregoing evolution rule is reduced to testing Π_d, while increasing the query complexity by a factor of $O(\log k) = O(\log n)$. Using the $O(d^2/\epsilon)$-query adaptive tester for Π_d (and observing that all queries can be made to any time in the second stage), Part 1 of Theorem 4.2 follows.

To prove Part 2 of the theorem, we reduce the task of testing Π_d to the task of testing validity of the evolving environments, while preserving the non-adaptivity of the tester. Specifically, given a tester T for environments, we obtain a tester T' for Π_d by invoking T (using a proximity parameter that is a constant times smaller than that provided to T') and emulating an environment that corresponds to our own input. Specifically, if T asks for a location (in the second stage) that should contain a bit in the encoding of some vertex, then T' queries its own input-oracle for the identity of this vertex, and return the adequate bit. But T may also query locations in the first stage, and in this case T' answers as follows. If the probed location corresponds to a bit that is part of the list of some vertex (either a bit held in a specific area or a bit in a list that is being transmitted in some step), then T' just answers by making the corresponding query to its own oracle (very much as we handled the queries to the second stage). If the probed location corresponds to a bit that is supposed to hold the current value of the flag bit of some vertex, then T' answer 1 (although this answer may not be consistent with a valid evolution), and it does so without making any queries.

When analyzing the behavior of T', we first note that when T' is given oracle access to a graph in Π_d, then the answers that T' provides T are consistent with a valid evolution (which produces a representation of this graph). This holds also with respect to the values returned for the flag bits, since in this case the flags would have retained the value 1 during such an evolution. Hence, T accepts (with probability at least 2/3), and so does T'.

Turning to the case that T' is given access to a graph that is ϵ-far from Π_d, we note that the answers it provides for queries in the second stage correspond to an array that is $\Omega(\epsilon)$-far from a valid evolution. This is the case since a valid evolution produces either a representation of a graph in Π_d or a dummy representation, whereas the answers provided by T' correspond to a representation of a graph that is ϵ-far from Π_d. Furthermore, the error correcting code used in these representations (on the level of individual vertices) preserves the distance between graphs. (Note that we did not capitalize on the fact that the flag bits are also inconsistent with a valid evolution.) Assuming that T was invoked with proximity parameter $\Omega(\epsilon)$, it follows that T rejects with probability at least 2/3, and so does T'. Part 2 of Theorem 4.2 follows. ■

4.3 On the effect of the temporal complexity

In this section we show that restricting the temporal query complexity of testers may cause an increase in their total query complexity. This is proved while relying on a partial viewing function, leaving the case of fully viewing function open.

Theorem 4.3 (on the effect of temporal complexity): *There exist a constant $c > 0$, a viewing function $V : \Sigma \to \Sigma'$ and $\Xi \subset \Sigma$, such that for any polynomially computable function $q : \mathbb{N} \to \mathbb{N}$ that satisfies $q(n) < n^c$, there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ for which the following holds.*

1. *Evolution from Ξ^* according to Γ via V can be tested by a* (time-conforming) *oracle machine that makes $O(\log n)^2 \cdot q(n) + O(1/\epsilon)$ queries, and*
2. *Evolution from Ξ^* according to Γ cannot be tested by an oracle machine of temporal query complexity $q(n)$ that makes n^c queries.*

Proof: We consider an evolution that proceeds in three stages, where the third stage occupies most of the time. In the first stage, which is performed using states that are totally hidden by the viewing function, the evolution generates $i \in \mathbb{Z}_k$, and a sequence of $t(n)$ numbers $r_1, ..., r_{t(n)} \in \mathbb{Z}_k$ (where $\mathbb{Z}_k = \{0, 1, ..., k-1\}$), where $t(n) = q(n) \cdot \log_2 n$ and $k = n^{3c}$. Both i and the r_j's are generated based on the hidden part of the states of the initial configuration, which in the analysis we shall consider to be random; hence, we may think of i and r_j's as being uniformly distributed in $\mathbb{Z}_k$. Likewise, the evolution generates random $u \in \{0,1\}^i$ and $v \in \{0,1\}^{k-i}$, and lets $x = uuvv$. We stress that during the entire first stage nothing is visible (i.e., the visible part of the states equals zero).

In the second stage, which lasts for $\log_2 n$ units of time, the evolution rule presents the sequence of values $(r_1, ..., r_{t(n)}, s)$, where $s = i + \sum_{j \in [t(n)]} r_j \bmod k$. Note that this sequence constitutes a $(t(n)+1)$-out-of-$(t(n)+1)$ secret sharing of i (i.e., i can be recovered when given all the $t(n)+1$ elements (called shares), but any set of $t(n)$ elements (shares) yields no information about i). Implementing this step requires synchronizing all cells of the automaton up to a deviation of $\log_2 n$ units, which can be performed by maintaining clocks in disjoint $\log_2 n$-long blocks (see Appendix A.1 for details). Next, in the third stage, which lasts for most of the time, the evolution rule presents and maintains x.

An obvious way of testing whether an evolution is consistent with Γ is to (1) test that during the first stage the state equals zero, (2) obtain i by reading all $t(n)+1$ values of the sequence presented in the second stage, and (3) use i to check if the string x presented in the third stage has the form $aabb$, where $|a| = i$. Hence, this tester has query complexity $(t(n)+1) \cdot \log_2 k + O(1/\epsilon) = O(\log n)^2 \cdot q(n) + O(1/\epsilon)$, and Part 1 follows.

In contrast, a machine of temporal query complexity $q(n)$ cannot reveal any information about i, which is presented only at few (i.e., $\log_2 k$) time units using a $(t(n)+1)$-out-of-$(t(n)+1)$ secret sharing scheme, whereas such a machine can only obtain $q(n) \cdot \log_2 n < t(n)+1$ shares. Hence, such a machine is faced with the problem of testing whether x belongs to the set $S = \{aabb \in \{0,1\}^{2k} : a, b \in \{0,1\}^*\}$, which (by [1]) requires query complexity $\Omega(\sqrt{k}) > n^c$.

Formally, testing the property S is reducible to testing whether an environment evolves according to Γ (via V): Given a tester T for environments, we obtain a tester T' for the property S by invoking T and answering its queries as follows. Queries to the first stage are answered by zero, queries to the second stage are answered by random values, and queries to the third stage are answered by making the corresponding queries to the oracle of T'. Part 2 follows. ■

Digest. Indeed, we capitalize on the fact that testing the property S has high query complexity, whereas verifying membership in it when given a short proof can be done using few queries; see [15] for a general perspective on this gap. Our argument relied on a viewing function that hides some information, and we wonder whether an analogous result holds for fully visible states.

Open Problem 4.4 (the effect of temporal complexity when the states are fully visible): *Does a result analogous to Theorem 4.3 hold for the case of fully visible states* (i.e., $V(\sigma) = \sigma$ for all σ)*?*

5 Linear Rules

In this section we consider the evolution of the environment under an arbitrary d-dimensional *linear rule* $\Gamma : F^{3^d} \to F$, where F is a finite field (and $d \geq 1$ (e.g., $d \in \{1,2,3\}$)); that is, we have

$$\Gamma(z_{-1^d}, .., z_{1^d}) = \sum_{\sigma \in \{-1,0,+1\}^d} \alpha_\sigma z_\sigma, \tag{3}$$

where the α_σ's are in F. Note that no generality is gained by allowing also affine rules, since one can reduce the study of affine rules to the study of linear rules by subtracting appropriate multiples of the corresponding free-term from the state of the environment.

We shall focus on the case that the states are fully visible; that is, assume that $V : \Sigma \to \Sigma'$ is the identity mapping (i.e., $V(\sigma) = \sigma$ for every $\sigma \in \Sigma$). Hence, in the rest of this section we typically do not mention V.

A kind of introduction. Given that Γ is a linear function, it follows that each entry in $\texttt{ENV}$ is a linear expression in the entries of $\texttt{ENV}_1$. Thus, testing reduces to sampling sufficiently many values, viewing each as a linear equation in the variables that correspond to the entries of $\texttt{ENV}_1$, and checking consistency. Indeed, if consistency holds, then we can also find a solution to the corresponding system of equations, thus efficiently solving the learning problem. The question is how many values (or equations) do we need to sample in order to check consistency of the linear system and ditto for finding a solution to the system. In both cases, the answer depends on Γ.

Let us start by considering the very special case in which Γ depends on a single variable. In this case the testing question reduces to testing n^d disjoint arrays (and making sure that the elements in them are properly related). For example, if $\Gamma(z_{-1,-1,-1}, .., z_{1,1,1}) = 5z_{0,0,0}$, then for every $i_1, i_2, i_3 \in [n]$ we need to check that the values in $\texttt{ENV}(\cdot, i_1, i_2, i_3)$ are properly related (i.e., $\texttt{ENV}_j(i_1, i_2, i_3) = 5^{j-1} \cdot \texttt{ENV}_1(i_1, i_2, i_3)$ for every $j \in [t]$). Hence, in this case $O(1/\epsilon)$ samples would suffice for testing (i.e., we shall check $O(1/\epsilon)$ random equations of the foregoing type).[20] Note that even in this case $\Omega(n^3)$ queries are required for learning.

In general (for $t = O(n/\epsilon)$), any learning algorithm must make $\Omega(n^d)$ queries, and one can improve over this only in degenerate cases (i.e., when Γ is a constant function). This will be shown in Proposition 5.1. On the other hand, as indicated by the case of Γ that depends on a single variable, more query-efficient testers are possible in some cases. In fact, $o(n^d/\mathrm{poly}(\epsilon))$-query

[20] If all checks pass (with high probability), then for all but at most ϵ fraction of $(j, i_1, i_2, i_3) \in [t] \times [n]^3$ it holds that $\texttt{ENV}_j(i_1, i_2, i_3) = 5^{j-1}\texttt{ENV}_1(i_1, i_2, i_3)$, which means that $\texttt{ENV}$ is ϵ-close to being consistent with the evolution of $\texttt{ENV}_1$ according to Γ. In contrast, note that the more straightforward idea of checking $\texttt{ENV}_{j+1}(i_1, i_2, i_3) = 5\texttt{ENV}_j(i_1, i_2, i_3)$, for uniformly selected $(j, i_1, i_2, i_3) \in [t-1] \times [n]^3$, will not do (e.g., equality may be violated only at one j, whereas the corresponding $\texttt{ENV}$ may be very far from being consistent with the evolution of $\texttt{ENV}_1$ according to Γ).

testers exist for any linear Γ over a finite field F of prime order: See Theorem 1.9, which will be proved in Section 5.2.

Detour: A connection to locally testable codes. Rules $\Gamma : \Sigma^{3^d} \to \Sigma$ for which the environment can be tested in $o(n^d)$ queries (in total) yield weak cases of locally testable codes (cf. [9]), which may be non-trivial. We refer to the code that arises by mapping the initial global state (i.e., $\mathtt{ENV}_1$) to its entire evolution over time (i.e., $\mathtt{ENV}$). Such a code maps n^d symbols to $t \cdot n^d$ symbols, and it may be of interest only if its relative distance significantly exceeds $1/t$. The connection holds also for non-linear Γ (yielding non-linear codes), but seems more appealing in the linear case.

5.1 More on learning

As stated above, in general (i.e., for $t = O(n/\epsilon)$), any learning algorithm must make $\Omega(n^d)$ queries, and one can improve over this only in degenerate cases (i.e., when Γ is a constant function). This will be shown in Proposition 5.1, but before stating and proving the proposition, we note that for $t > n/\epsilon$ there exist non-degenerate rules for which the environment vanishes and so can be trivially learned (e.g., consider $\Gamma(z_{-1,...,-1}, .., z_{1,...,1}) = z_{1,...,1}$, which implies $\mathtt{ENV}_j \equiv 0^{n^d}$ for every $j > n$).

Proposition 5.1 *For any $t = O(n/\epsilon)$ and non-constant linear $\Gamma : F^{3^d} \to F$, any algorithm for learning environments that evolve according to Γ must make $\Omega(n^d)$ queries.*

Proof: For the sake of simplicity, we consider the case $t = n$; the argument generalizes for any $t = O(n/\epsilon)$. Let $\Gamma(z_{-1,...,-1}, .., z_{1,...,1}) = \sum_{s_1,...,s_d \in \{-1,0,1\}} c_{s_1,...,s_d} z_{s_1,...,s_d}$, and consider some $(s_1, ..., s_d) \in \{-1, 0, 1\}^d$ with a minimal number of zeros such that $c_{s_1,...,s_d} \neq 0$. The reader may find it instructive to think of the case that $(s_1, ..., s_d) \in \{-1, 1\}^d$ (or even $s_1 = \cdots = s_d = 1$). For every $i_1, ..., i_d \in [n]$, consider the j-parameterized line $L_{i_1,...,i_d}(j) = (j, i_1 + (j-1)s_1, ..., i_d + (j-1)s_d)$. Note that the value of $\mathtt{ENV}$ on this line depends on the value of $\mathtt{ENV}_1(i_1, ..., i_d)$. However, for every $i'_1, ..., i'_d \in [n]$ such that $s_k i'_k > s_k i_k$ holds for some $k \in [d]$, the value of $\mathtt{ENV}$ on the line $L_{i'_1,...,i'_d}$ does not depend on the value of $\mathtt{ENV}_1(i_1, ..., i_d)$. This is easiest to see when $(s_1, ..., s_d) \in \{-1, 1\}^d$, but in the general case one needs to use the hypothesis that $(s_1, ..., s_d) \in \{-1, 0, 1\}^d$ is minimal with respect to number of zeros (which implies that $c_{s'_1,...,s'_d} = 0$ for every $(s'_1, ..., s'_d) \neq (s_1, ..., s_d)$ such that $s'_i = s_i$ for every i with $s_i \neq 0$). The same can be proved for every $i'_1, ..., i'_d \in [n]$ such that $i'_k \neq i_k$ and $s_k = 0$ hold for some $k \in [d]$. It follows that the values of these n^d lines (i.e., the values at any set of n^d positions that reside on different lines), expressed as a linear combination of the values of $\mathtt{ENV}_1$, are linearly independent. Since a constant fraction $c > 0$ of the volume of $[n] \times [n]^d$ is covered by these n^d lines, a learning algorithm must query points on cn^d of these lines in order to infer the value of a random point in $[n] \times [n]^d$ with probability at least $1 - c/2$. ■

Tightness of Proposition 5.1. The lower bound stated in Proposition 5.1 is tight: Indeed, for any linear Γ (and the identity viewing function, representing fully visible states), we can obtain an efficient learning algorithm that makes $O(n^d/\epsilon t)$ queries to each $\mathtt{ENV}_j$. Let $\zeta_i \leq n^d$ be a random variable that denotes the number of independent linear expressions (in the $\mathtt{ENV}_1$-variables) that we see when we sample i random locations in $[t] \times [n]^d$, where each location corresponds to a linear expression. Letting $p(i)$ denote the probability that the $i + 1^{\text{st}}$ sample yields an expression that is linearly independent of the previous i (sampled) expressions, note that $\mathrm{E}[\zeta_{i+1} - \zeta_i] = p(i)$. Using

the linearity of expectation it follows that $\sum_{i=1}^{3n^d/\epsilon} p(i) \leq n^d$, and so $p(i) < \epsilon$ for at least a $2/3$ fraction of the $i \in [3n^d/\epsilon]$. Hence, the learning problem may be solved by picking i uniformly in $[3n^d/\epsilon]$, making i uniformly selected queries to $\texttt{ENV}$, and solving the corresponding linear system. (Indeed, if $p(i) < \epsilon$, then the solution found will fit at least a $1 - \epsilon$ fraction of the domain of $\texttt{ENV}$.)

The above ideas may be applied also in the case that the states are not fully visible, provided that the viewing function V is linear when Σ is an extension field of some smaller field (and V is linear over the small field). All that is needed is to apply the foregoing considerations to the smaller field; that is, note that each value in $V \circ \texttt{ENV}$ is a linear combination (over the small field) of the values in the sequences in $\texttt{ENV}_1$, where each element of $\texttt{ENV}_1$ is viewed as a sequence of elements in the small field.

5.2 Testing is easier than learning

In this section we prove Theorem 1.9, which asserts that *for any $d \geq 1$ and any linear $\Gamma : \Sigma^{3^d} \to \Sigma$, there exists a constant $\gamma < d$ and an oracle machine of total time complexity $\mathrm{poly}(\epsilon^{-1}) \cdot n^\gamma$ that tests the consistency of evolving environments with $\Gamma : \Sigma^{3^d} \to \Sigma$ and the identity mapping.* Hence, for any non-constant linear rule Γ, testing evolution according to Γ is easier than learning this evolution.

We shall proceed in three steps, starting from the special case of $d = 1$ and $|F| = 2$, moving to a general finite field F (of prime order, still with $d = 1$), and finally treating any $d \geq 1$. Recall that we already showed (at the beginning of Section 5) that evolution according to a linear rule that depends on a single variable can be tested using $O(1/\epsilon)$ queries.

A generic tester for one-dimensional environments. The following tester refers to environments that are supposed to be determined by a linear rule $\Gamma : F^3 \to F$. Actually, we shall present a proximity-oblivious tester (cf. [13]), which rejects any environment that is ϵ-far from being consistent with Γ with probability at least $\epsilon/2$. A standard tester can be derived by invoking this proximity-oblivious tester $O(1/\epsilon)$ times. The proximity-oblivious tester depends on a (constant) parameter γ, which will be determined later (e.g., for $|F| = 2$ we may use $\gamma = 0.8$). On oracle access to $\texttt{ENV} : [t] \times [n] \to F$, the tester proceeds as follows:

1. The tester selects $(j, i) \in ([t] \times [n])$ uniformly at random;
2. If location (j, i) in $\texttt{ENV}$ is determined by at most $2n^\gamma/\epsilon$ locations in the first row (i.e., $\texttt{ENV}_1$), then the tester queries these locations in $\texttt{ENV}_1$ and accepts iff their value fit $\texttt{ENV}(j, i)$, which it queries too. Otherwise, the tester accepts without making any queries.

In the analysis we assume for simplicity that $t = n$; the general case will be handled at the very end of this section. Clearly, if $\texttt{ENV}$ is consistent with Γ, then the tester accepts with probability 1. Suppose that $\texttt{ENV}$ is ϵ-far from being consistent with Γ. Then, more than an ϵ fraction of the locations in $\texttt{ENV}$ are inconsistent with the Γ-evolution of the first row of $\texttt{ENV}$. As we shall show (in Claim 5.4 below), the expected number of locations in the first row that determine a random location in $\texttt{ENV}$ is at most n^γ. Therefore, the probability that $\texttt{ENV}(j, i)$ depends on more than $2n^\gamma/\epsilon$ locations in the first row of $\texttt{ENV}$ is smaller than $\epsilon/2$. It follows that *the tester rejects $\texttt{ENV}$ with probability at least $\epsilon - \epsilon/2$.*

Fixing the finite field F and the linear rule Γ, let us denote by $D_i(t_0, \ell)$ the set of locations in time t_0 that influence the value of a specific location ℓ in time $t_0 + i$. Note that $|D_i(t_0, \ell)|$ is invariant

under t_0 and ℓ, and, since we only care of the cardinality of $D_i(t_0,\ell)$, we allow ourselves to omit (t_0,ℓ) from the notation. Also note that $\frac{1}{n}\sum_{i=0}^{n-1}|D_i(1,\cdot)|$ equals the expected number of locations in the first row that determine a random location in `ENV`, which is the quantity that governs the query complexity of our tester.

Step 1: The binary field. As a warm-up, we first consider the binary field and the rule $\Gamma(z_{-1},z_0,z_1)=z_{-1}+z_0+z_1 \bmod 2$. The reader may verify that better bounds can be obtained for the other linear rules (over the binary field).[21]

Claim 5.2 (warm-up): *For $\Gamma(z_{-1},z_0,z_1)=z_{-1}+z_0+z_1 \bmod 2$, it holds that $\sum_{i=1}^{n}|D_i|<n^{1.8}$.*

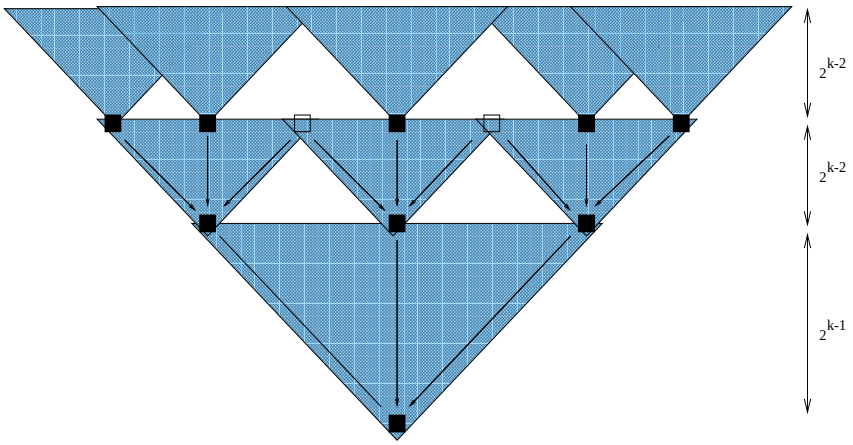


Figure 4: *Demonstrating that $A_k < A_{k-1}+8A_{k-2}$ (for the proof of Claim 5.2). Only the shaded triangles contain locations that may influence the value of the bottom location.*

Proof: It will be most convenient to consider $D_i = D_i(0,0)$, which means that we consider an infinite table $T:\mathbb{Z}\times\mathbb{Z}\to\{0,1\}$ (with locations and time units that are associated with all integers); that is, $T(t',\ell)$ represents the contents of location ℓ at time t'. Hence, for every $i\in\mathbb{N}$, it holds that $T(t',\ell)$ equals $\sum_{\ell'\in D_i}T(t'-i,\ell+\ell')$. Again, it will be convenient to let $T_{t'}(\ell)=T(t',\ell)$. Note that $D_1=\{-1,0,+1\}$, and we shall show (by induction) that for every $i>0$ it holds that $D_{2^i}=\{-2^i,0,+2^i\}$. Using mod 2 arithmetic, in the induction step, we have

$$\begin{aligned}
T_0(0) &= \sum_{\ell\in D_{2^{i-1}}} T_{-2^{i-1}}(\ell)\\
&= T_{-2^{i-1}}(-2^{i-1})+T_{-2^{i-1}}(0)+T_{-2^{i-1}}(2^{i-1})\\
&= \sum_{\ell\in D_{2^{i-1}}} T_{-2^i}(-2^{i-1}+\ell)+\sum_{\ell\in D_{2^{i-1}}} T_{-2^i}(\ell)+\sum_{\ell\in D_{2^{i-1}}} T_{-2^i}(2^{i-1}+\ell)\\
&= \left(T_{-2^i}(-2^i)+T_{-2^i}(-2^{i-1})+T_{-2^i}(0)\right)
\end{aligned}$$

[21] Recall that we already showed that evolution according to a linear rule that depends on a single variable can be tested using $O(1/\epsilon)$ queries. For linear rules that depend on two variables, one can first show that $|D_{2^i}|=2$ for every $i\in\mathbb{N}$. Next, denoting $A_k=\sum_{i\in[2^k]}|D_i|$, one can show that $A_k\le A_{k-1}+|D_{2^{k-1}}|\cdot A_{k-1}$, and it follows that $A_k\le 3A_{k-1}\le 3^k$, which implies that $A_k\le(2^k)^{\log_2 3}$. The proof of Claim 5.2 is merely a more refined analysis of a similar nature.

$$
\begin{aligned}
&+ \left(T_{-2^i}(-2^{i-1}) + T_{-2^i}(0) + T_{-2^i}(2^{i-1})\right) \\
&+ \left(T_{-2^i}(0) + T_{-2^i}(2^{i-1}) + T_{-2^i}(2^i)\right) \\
= \;& T_{-2^i}(-2^i) + T_{-2^i}(0) + T_{-2^i}(2^i),
\end{aligned}
$$

which establishes $D_{2^i} = \{-2^i, 0, +2^i\}$.

Now, let $A_k \stackrel{\text{def}}{=} \sum_{i=1}^{2^k} |D_i|$. Using induction, we shall show that $A_k < 2 \cdot (2^k)^{1.76}$. The base case (i.e., $k = 1, 2$) follows since $A_1 = 3 + 3 = 6 < 2 \cdot 2^{1.76}$ and $A_2 = A_1 + 5 + 3 = 14 < 2 \cdot 2^{2 \cdot 1.76}$.

For the inductive step we use $A_k < A_{k-1} + 8A_{k-2}$, which is proved by writing $A_k = A_{k-1} + \sum_{i=2^{k-1}+1}^{2^k} |D_i|$ and looking closely at the latter sum (see Figure 4). Specifically, Figure 4 shows regions that contain locations in $\cup_{i \in [2^k]} D_i$ with respect to the bottom location. The key observation is that $D_{2^{k-1}}$ contains only three locations and that $D_{2^{k-1}+i}$ is contained in $D_{2^{k-1}} + D_i = \{\ell + \ell' : \ell \in D_{2^{k-1}}, \ell' \in D_i\}$, which implies $|D_{2^{k-1}+i}| \leq 3 \cdot |D_i|$. Furthermore, for $i > 2^{k-2}$, it holds that $D_{2^{k-1}+i}$ is contained in $D_{2^{k-1}+2^{k-2}} + D_{i-2^{k-2}}$, which implies $|D_{2^{k-1}+i}| \leq |D_{2^{k-1}+2^{k-2}}| \cdot |D_{i-2^{k-2}}|$. A non-crucial improvement (which relies on the specific linear rule analyzed here) is obtained by noting that $D_{2^{k-1}+2^{k-2}}$ contains five locations (rather than seven).[22] Hence,

$$
\begin{aligned}
A_k &= \sum_{i=1}^{2^{k-1}} |D_i| + \sum_{i=1}^{2^{k-2}} |D_{2^{k-1}+i}| + \sum_{i=2^{k-2}+1}^{2^{k-1}} |D_{i-2^{k-1}}| \\
&\leq A_{k-1} + \sum_{i=1}^{2^{k-2}} 3 \cdot |D_i| + \sum_{j=1}^{2^{k-2}} 5 \cdot |D_j| \\
&= A_{k-1} + (3+5) \cdot A_{k-2}
\end{aligned}
$$

follows. The crucial point is that $A_k < A_{k-1} + c \cdot A_{k-2}$, for some $c < 12$.

Now, combining $A_k < A_{k-1} + 8A_{k-2}$ with the induction hypothesis (i.e., $A_{k'} < 2 \cdot (2^{k'})^{1.76}$ for $k' < k$), we get $A_k/2 < (2^{k-1})^{1.76} + 8 \cdot (2^{k-2})^{1.76}$, and the induction step is completed since $(2^{k-1})^{1.76} + 8 \cdot (2^{k-2})^{1.76} < (2^k)^{1.76}$, which holds since $2^{1.76} + 8 < 4^{1.76}$. ■

Remark 5.3 *For Γ as in Claim 5.2, it holds that $\sum_{i=1}^{n} |D_i| > n^{1.5}$. This holds since $A_k > A_{k-1} + (4 + 4 \cdot 0.5) \cdot A_{k-2}$, which implies $A_k > 3^k$.*

Step 2: Extending the argument to any finite field of prime order. Staying with the one-dimensional case, we now turn to the general case of a linear rule over a finite field (of prime order). Throughout the rest of this section, the arithmetics is that of the finite field.

Claim 5.4 (the one-dimensional case): *For $\Gamma(z_{-1}, z_0, z_1) = \alpha_{-1} z_{-1} + \alpha_0 z_0 + \alpha_1 z_1$, where the arithmetics is of the finite field F of prime order p, there exists $\gamma < 1$ such that $\sum_{i=1}^{n} |D_i| < n^{1+\gamma}$.*

Proof: We first prove the following fact.

Fact 5.4.1 *For $i = 1, ..., p$, and every integer ℓ, let*

$$
x_\ell^{(i)} = \alpha_{-1} x_{\ell-1}^{(i-1)} + \alpha_0 x_\ell^{(i-1)} + \alpha_1 x_{\ell+1}^{(i-1)} . \tag{4}
$$

Then, it holds that $x_\ell^{(p)} = \alpha_{-1} x_{\ell-p}^{(0)} + \alpha_0 x_\ell^{(0)} + \alpha_1 x_{\ell+p}^{(0)}$.

[22] This improvement is non-crucial given that we already established that, for $i \in [2^{k-2}]$, it holds that $|D_{2^{k-1}+i}|$ is bounded by three times the size of D_i (rather than five times that amount).

Using Fact 5.4.1, we get $D_1 \subseteq \{-1,0,+1\}$ and for every $k > 0$ it holds that $D_{p^k} = p^k \cdot D_1$ (by induction on k): Both facts are proved by letting $x^{(i)}_\ell = T_{\ell \cdot p^{k-1}}((i-p) \cdot p^{k-1})$ (where T is as defined in the proof of Claim 5.2), and using Fact 5.4.1. We now turn to prove Fact 5.4.1.

Proof: Applying straightforward recursive substitution to Eq. (4), we get

$$\begin{aligned} x^{(p)}_\ell &= \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \cdot x^{(0)}_{\ell+\sigma_1+\cdots+\sigma_p} \\ &= \sum_{\tau \in [-p,p]} \mathtt{d}(\tau) \cdot x^{(0)}_{\ell+\tau} \qquad (5) \\ \text{where } \mathtt{d}(\tau) &\stackrel{\text{def}}{=} \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}:\ \Sigma_i \sigma_i = \tau} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \qquad (6) \end{aligned}$$

We start by analyzing Eq. (6). Letting $S_p(\tau) \stackrel{\text{def}}{=} \{(\sigma_1,...,\sigma_p) \in \{-1,0,+1\}^p : \sum_i \sigma_i = \tau\}$, note that $S_p(p)$ (resp., $S_p(-p)$) contains only the all-one (resp., all-minus-one) sequence. Thus, for every $\sigma \in \{\pm 1\}$, we have $\mathtt{d}(\sigma \cdot p) = \alpha_\sigma \cdots \alpha_\sigma = \alpha^p_\sigma$, which equals $\alpha_\sigma \pmod p$. This handles the case of $\tau \in \{-p,p\}$. Turning to the case of $\tau \in [-(p-1),(p+1)]$, let $S_p(\tau,j)$ denote the sequences in S_p that have exactly j zero-entries. Then, for $\tau \in [-(p-1),(p+1)]$, we have

$$\begin{aligned} \mathtt{d}(\tau) &= \sum_{(\sigma_1,...,\sigma_p) \in S_p(\tau)} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \\ &= \sum_{j=0}^{p} |S_p(\tau,j)| \cdot \alpha^j_0 \cdot \alpha_1^{(p-j+\tau)/2} \cdot \alpha_{-1}^{(p-j-\tau)/2} \qquad (7) \end{aligned}$$

Note that $|S_p(\tau,j)| = \binom{p}{j} \cdot \binom{p-j}{(p-j+\tau)/2}$ if $p-j-|\tau|$ is non-negative and even, and $S_p(\tau,j)$ is empty otherwise. Also, for every $j \in [p-1]$, it holds that $\binom{p}{j} \equiv 0 \pmod p$, whereas for $j = 0$ (and $\tau \in [-(p-1),(p-1)]$ such that $p-|\tau|$ is even) it holds that $\binom{p-j}{(p-j+\tau)/2} \equiv 0 \pmod p$. Hence, for every $\tau \in [-(p-1),(p-1)]$, the sum in Eq. (7) contains only the term that corresponds to $j = p$, which is zero if $\tau \neq 0$ (since in that case $p-j-|\tau|$ is negative), and equals $\alpha_0 \pmod p$ otherwise (since in that case (i.e., $\tau = 0$), it holds that $\mathtt{d}(0) \equiv |S_p(0,p)| \cdot \alpha^p_0 \equiv \alpha_0 \pmod p$). It follows that $\mathtt{d}(0) = \alpha_0$, whereas $\mathtt{d}(\tau) = \mathtt{d}(-\tau) = 0$ for every $\tau \in [p-1]$. Recalling that $\mathtt{d}(p) = \alpha_1$ and $\mathtt{d}(-p) = \alpha_{-1}$, and plugging everything into Eq. (5), the fact follows. ■

Recall that Fact 5.4.1 implies that, for every $k \geq 0$, it holds that $D_{p^k} \subseteq \{-p^k, 0, p^k\}$. Now, let $A_k = \sum_{i=1}^{p^k} |D_i|$. Using induction, we shall show that there exists $\beta < 2$ such that $A_k < 2p^{4+\beta \cdot (k-2)}$. The base case (i.e., $k = 1,2$) is trivial, since $A_k \leq \sum_{i=1}^{p^k}(2i+1) < 2p^{2k}$ always holds. We next show that

$$\begin{aligned} A_k &\leq A_{k-1} + \left(3 + \sum_{j=2}^{p}(2p+2j-1)\right) \cdot A_{k-2} + \sum_{j=3}^{p}(2j-1) \cdot A_{k-1} \\ &= (p^2-3) \cdot A_{k-1} + (3p^2-2p+2) \cdot A_{k-2} \end{aligned}$$

where the inequality is shown in Figure 5: In the figure, large triangles (of height p^{k-1}) are used for all levels except the second one, where smaller triangles (of height p^{k-2}) are used, and the saving

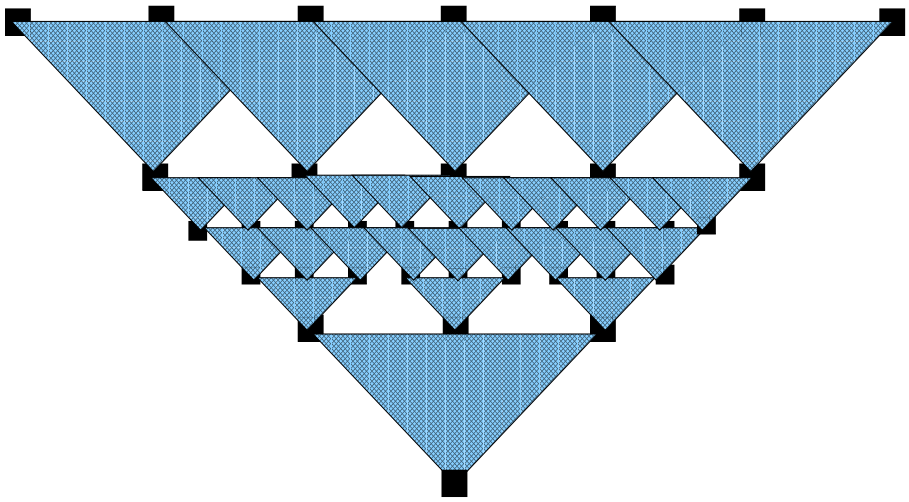


Figure 5: *The case of $p = 3$ (for the proof of Claim 5.4). Smaller triangles replace the three big triangles that would have appeared in the second level. As in Figure 4, only the shaded triangles contain locations that may influence the value of the bottom location.*

comes from there (i.e., from the second level).[23] In the first sub-level of the second level, three triangles were used (rather than $2p+1$), whereas in the j^{th} sub-level $2p+2j-1$ triangles were used (for any $j > 1$). Thus, the total number of small triangles used is $3+\sum_{j=2}^{p}(2p+2j-1) = 3p^2-2p-2$. The number of large triangles used is $1 + \sum_{j=3}^{p}(2j-1) = p^2 - 3$, since $2j-1$ large triangles are used in the j^{th} level.

To complete the argument, we set $\beta < 2$ such that $(p^2-3)\cdot p^{4+\beta\cdot(k-3)} + (3p^2-2p+2)\cdot p^{4+\beta\cdot(k-4)}$ is smaller than $p^{4+\beta\cdot(k-2)}$. This is possible since there exists an $X < p^2$ such that $(p^2-3)\cdot X + (3p^2-2p+2) < X^2$ (and setting $\beta = \log_p X$ we are done).[24] ■

Remark 5.5 *We comment that for Γ as in Claim 5.4, if $|D_1| > 1$, then there exists $\gamma > 0$ such that $\sum_{i=1}^{n}|D_i| > n^{1+\gamma}$. For $|D_1| = 3$, this holds since $A_k > A_{k-1} + \sum_{j=2}^{p}(2j-1)\cdot A_{k-2}$, where the A_{k-1} is due to the large triangle at the first level of Figure 5 and the A_{k-2}'s are due to the small triangles that appear in the first sub-level of each other level. Actually, some of these small triangles may not be influential, which happens when the location of their head is not influential, but in this case we may compensate by the area of some of the large triangles in this level, which is much larger* (i.e., $|A_{k-1}| > p\cdot|A_{k-2}|$). *Hence, $A_k > A_{k-1} + (p^2-1)\cdot A_{k-2}$, and $A_k > p^{(1+\Omega(1))\cdot k}$ follows. For $|D_1| = 2$, the argument is even simpler; we may rely solely on the A_{k-1}'s, and obtain $A_k \geq (2p-1)\cdot A_{k-1}$.*

[23] In general, we have p levels such that for every $j \in [p]\setminus\{2\}$ we use $2j-1$ large triangles in the j^{th} level. Figure 4 corresponds to the case of $p = 2$ and so a large triangle was used only in the first level. Recall that additional saving was obtained in Figure 4 by using five small triangles (rather than seven) in the second sub-level of the second level. This cannot be done (and is not done) in Figure 5.

[24] The equation $X^2 - (p^2-3)X - (3p^2-2p+2) = 0$ has one negative solution and one solution in the interval $(0, p^2)$, since $2X = (p^2-3) \pm \sqrt{(p^2-3)^2 + 4\cdot(3p^2-2p+2)}$ and $(p^2-3)^2 + 4\cdot(3p^2-2p+2) = (p^2+3)^2 - 8(p-1)$. For $p = 3$, the positive solution is $X \approx 8.657$, and $\log_3 X \approx 1.965$.

Step 3: Extending the argument to any $d \geq 1$. Finally, we turn to the d-dimensional case, for $d \geq 2$ (e.g., $d = 2, 3$). Firstly, we generalize the definition of D_i such that $D_i \subset \mathbb{Z}^d$ is the set of locations in time $-i$ that influence location 0^d in time 0.

Claim 5.6 (the d-dimensional case): *Let $d \geq 2$ and $\Gamma : F^{\{-1,0,+1\}^d} \to F$ such that $\Gamma(z_{-1^d}, ..., z_{1^d}) = \sum_{\sigma \in \{-1,0,+1\}^d} \alpha_\sigma z_\sigma$, where the arithmetics is of the finite field F of prime order p. Then, there exists $\gamma < d$ such that $\sum_{i=1}^n |D_i| < n^{1+\gamma}$.*

By a straightforward generalization of the algorithm presented for the one-dimensional case, Claim 5.6 yields an algorithm that establishes Theorem 1.9.

Proof: We generalize the proof of Claim 5.4. Here for $i = 1, ..., p$ and every d-dimensional location $v \in \mathbb{Z}^d$, we consider

$$x_v^{(i)} = \sum_{\sigma \in \{-1,0,+1\}^d} \alpha_\sigma x_{v+\sigma}^{(i-1)} \tag{8}$$

As in the proof of Fact 5.4.1, we get

$$\begin{aligned} x_v^{(p)} &= \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}^d} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \cdot x_{v+\sigma_1+\cdots+\sigma_p}^{(0)} \\ &= \sum_{\tau \in [-p,p]^d} \mathtt{d}(\tau) \cdot x_{v+\tau}^{(0)} \\ \text{where } \mathtt{d}(\tau) &\stackrel{\text{def}}{=} \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}^d:\ \Sigma_i \sigma_i = \tau} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \end{aligned}$$

Let $\Delta \stackrel{\text{def}}{=} \{-1, 0, +1\}^d$. For an arbitrary p-long sequence $(\sigma_1, ..., \sigma_p) \in \Delta^p$, we let $P(\sigma_1, ..., \sigma_p)$ denote the set of sequences that are permutations of the sequence $(\sigma_1, ..., \sigma_p)$. For any uniform sequence $\sigma^p \in \Delta^p$, it holds that $|P(\sigma^p)| = 1$. The key observation is that for any non-uniform sequence $(\sigma_1, ..., \sigma_p)$, the size of $P(\sigma_1, ..., \sigma_p)$ is a multiple of p (since $|P(a^i b^{p-i})| = \binom{p}{i}$ which is a multiple of p for any $i \in [p-1]$). Observing that a non-uniform sequence $(\sigma_1, ..., \sigma_p)$ contributes to $\mathtt{d}(\tau)$ if and only if each element in $P(\sigma_1, ..., \sigma_p)$ contributes to $\mathtt{d}(\tau)$, it follows that the contribution of non-uniform p-long sequences over Δ to $\mathtt{d}(\tau)$ cancels out modulo p (since this contribution comes in multiples of p). If follows, that only the uniform sequences contribute to $\mathtt{d}(\tau) \pmod p$, which means that

$$\mathtt{d}(\tau) \equiv \sum_{\sigma \in \Delta:\ p \cdot \sigma = \tau} \alpha_\sigma^p \pmod p,$$

which in turn means that the sum has at most one element (i.e., the element corresponding to τ/p, where $\tau \in [-p, p]^d$). Hence, for every $\tau \in [-p, p]^d \setminus \{-p, 0, p\}^d$, it holds that $\mathtt{d}(\tau) = 0$ (since τ is not in the set $p \cdot \Delta$), whereas for every $\tau = p \cdot \sigma$ with $\sigma \in \Delta$ it holds that $\mathtt{d}(\tau) = \alpha_\sigma^p$ (and $\alpha_\sigma^p \equiv \alpha_\sigma \pmod p$ follows, since $p = |F|$). Thus, for every $k \geq 0$, it holds that $D_{p^k} \subseteq \{-p^k, 0, p^k\}^d$.

Again, we let $A_k = \sum_{i=1}^{p^k} |D_i|$, and proceed to upper bound A_k. We first consider the case of $p = 2$.

Fact 5.6.1 (the case of $p = 2$): *There exists $\beta < d + 1$ such that $|A_k| = O(2^{\beta k})$.*

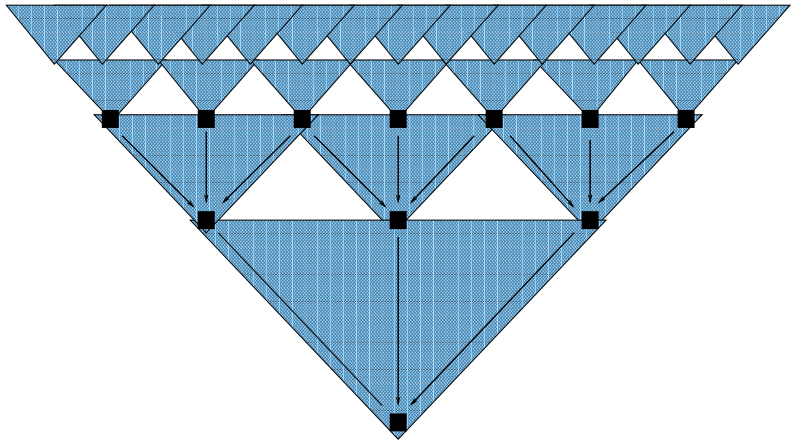


Figure 6: *A one-dimensional projection of the case of $p = 2$ and $k' = 3$ (for the proof of Claim 5.6). Only the shaded triangles contain locations that may influence the value of the bottom location.*

Proof: Generalizing the argument in Claim 5.2, we observe that for any fixed $k' \in [k-1]$ it holds that

$$A_k \leq \sum_{i \in [k']} (2^i - 1)^d \cdot A_{k-i} + (2^{k'+1} - 1)^d \cdot A_{k-k'} \,,$$

where Claim 5.2 used $d = 1$ and $k' = 2$ (with an extra observation that allowed us to replace 7 by 5) and the current observation is illustrated in Figure 6 (e.g., for $d = 1$ and every $i \in [k']$, we cover the i^{th} layer (which has height 2^{k-i}) with $2^i - 1$ triangles of height 2^{k-i}, and the remaining area is covered by $2^{k'+1} - 1$ triangles of height $2^{k-k'}$). Thus, to establish the inductive step (by which $|A_k| = O(2^{\beta k})$ for some $\beta < d+1$), we should show that $\sum_{i \in [k']} (2^i - 1)^d \cdot 2^{-i\beta} + (2^{k'+1} - 1)^d \cdot 2^{-k'\beta} < 1$. Using $\beta > d$, we get

$$\begin{aligned} \sum_{i=1}^{k'} (2^i - 1)^d \cdot 2^{-i\beta} + (2^{k'+1} - 1)^d \cdot 2^{-k'\beta} &< 2^{-\beta} + \sum_{i=2}^{k'} 2^{id} \cdot 2^{-i\beta} + 2^{(k'+1)d} \cdot 2^{-k'\beta} \\ &= 2^{-\beta} + \sum_{i=2}^{k'} 2^{-(\beta-d)\cdot i} + 2^{-(\beta-d)k'+d} \,. \end{aligned}$$

Picking $\beta \approx d + 1$, the last expression is approximately $2^{-(d+1)} + (0.5 - 2^{-k'}) + 2^{d-k'}$, which is strictly smaller than 1 provided that $d \geq 1$ and $k' \geq d + 2$. Hence, the induction claim follows. ■

Having established the claim for the case of $p = 2$, we now turn to the case of $p > 2$, where we shall use a more refined analysis (which builds on the same ideas).

Fact 5.6.2 (the case of $p > 2$): *There exists $\beta < d + 1$ such that $|A_k| = O(2^{\beta k})$.*

Proof: Fixing any $p > 2$, for every m, we let $A_k^{(m)} = \sum_{i=1}^{p^k} |\cup_{j_1,...,j_d \in \{0,...,m-1\}} (\{p^k \cdot (j_1, ..., j_d)\} + D_i)|$; that is, $A_k^{(m)}$ is the sum of the locations at times $\{-i : i \in [p^k]\}$ that influence any of the locations $\{(j_1 p^k, ..., j_d p^k) : j_1, ..., j_d \in \{0, ..., m-1\}\}$ at time 0. Indeed, $A_k^{(1)}$ is identical to A_k, whereas $\{(j_1, ..., j_d) \cdot p^k\} + D_i$ intersects $\{(j'_1, ..., j'_d) \cdot p^k\} + D_i$ only if $i > p^k/2$ (unless, of course, $(j_1, ..., j_d) =$

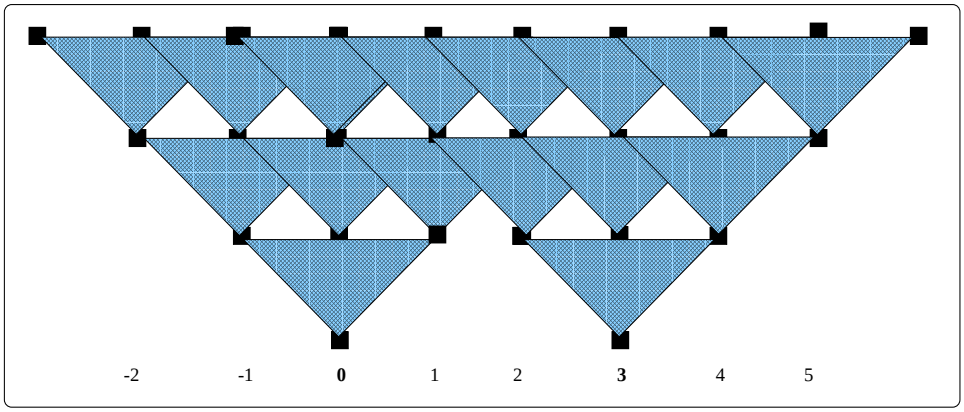


Figure 7: *A one-dimensional projection of the case of $p = 3$ and $m = 2$ (for the proof of Claim 5.6). The marked positions are scaled by 3^{k-1}. Only the shaded triangles contain locations that may influence the values of the $m = 2$ bottom locations. The bottom level represents 2^d copies of A_{k-1}, the middle level represents a copy of $A^{(6)}_{k-1}$, whereas the top level represents a copy of $A^{(8)}_{k-1}$.*

$(j'_1, ..., j'_d)$). Abusing notation such that $A^{(m)}_k$ denote a monotonic in m upper bound on $A^{(m)}_k$, the key observation is that

$$A^{(m)}_k \leq m^d \cdot A_{k-1} + \sum_{j=1}^{p-1} A^{((m-1)p+2j+1)}_{k-1}\,, \tag{9}$$

where $(m-1)p + 2j + 1$ represents the number of integer multiples of p^{k-1} in the interval $[-j \cdot p^{k-1}, (m-1) \cdot p^k + j \cdot p^{k-1}]$; see Figure 7. (Note that the inequality in Eq. (9) is tight if we ignore cancelations that were not stated explicitly before.)[25] We also have $A^{(m)}_k \leq m^d \cdot A_k$, which is wasteful. Using $(m-1)p + 2j + 1 \leq (m-1)p + 2p - 1 < mp + p$ (for any $j \in [p-1]$), and letting $N(m) = mp + p$, we get

$$A^{(m)}_k \leq m^d \cdot A_{k-1} + (p-1) \cdot A^{(N(m))}_{k-1}\,. \tag{10}$$

Letting $N_0(m) = m$ and $N_i(m) = N(N_{i-1}(m))$, and using Eq. (10), we get

$$\begin{aligned}
A^{(m)}_k &\leq N_0(m)^d \cdot A_{k-1} + (p-1) \cdot A^{(N_1(m))}_{k-1} \\
&\leq N_0(m)^d \cdot A_{k-1} + (p-1) \cdot \left(N_1(m)^d \cdot A_{k-2} + (p-1) \cdot A^{(N_2(m))}_{k-2} \right) \\
&= N_0(m)^d \cdot A_{k-1} + (p-1) \cdot N_1(m)^d \cdot A_{k-2} + (p-1)^2 \cdot A^{(N_2(m))}_{k-2} \\
&\leq \left(\sum_{i=1}^{k'} (p-1)^{i-1} \cdot N_{i-1}(m)^d \cdot A_{k-i} \right) + (p-1)^{k'} \cdot A^{(N_{k'}(m))}_{k-k'}
\end{aligned}$$

Using $N_i(m) = p \cdot N_{i-1}(m) + p = p^i m + \sum_{j=1}^{i} p^j < 2.5mp^i$ (since $p \geq 3$), we get

$$A^{(m)}_k \leq \left(\sum_{i=1}^{k'} (p-1)^{i-1} \cdot (2.5mp^{i-1})^d \cdot A_{k-i} \right) + (p-1)^{k'} \cdot A^{(2.5mp^{k'})}_{k-k'}$$

[25] That is, if we only use $D_{p^i} = \{-p^i, 0, p^i\}$ and $D_{i+1} \subseteq D_i + D_1$.

$$\leq \left(\sum_{i=1}^{k'}(p-1)^{i-1}\cdot(2.5mp^{i-1})^d\cdot A_{k-i}\right)+(p-1)^{k'}\cdot(2.5mp^{k'})^d\cdot A_{k-k'}$$

where the second inequality uses $A^{(M)}_{k'}\leq M^d\cdot A_{k'}$. Now, in order to prove that there exists fixed c and $\beta<d+1$ (which may both depend on p and d) such that for all k it holds that $|A_k|<c\cdot p^{\beta k}$, we need to prove that

$$Q\stackrel{\text{def}}{=}\left(\sum_{i=1}^{k'}(p-1)^{i-1}\cdot(2.5p^{i-1})^d\cdot p^{-i\beta}\right)+(p-1)^{k'}\cdot(2.5p^{k'})^d\cdot p^{-k'\beta}<1\,.$$

Note that

$$\begin{aligned}Q &= \frac{2.5^d}{p^\beta}\cdot\sum_{i=1}^{k'}\Big((p-1)\cdot p^d\cdot p^{-\beta}\Big)^{i-1}+2.5^d\cdot\Big((p-1)\cdot p^d\cdot p^{-\beta}\Big)^{k'}\\ &= \frac{2.5^d}{p^\beta}\cdot\sum_{i=1}^{k'}q^{i-1}+2.5^d\cdot q^{k'}\end{aligned}$$

where $q\stackrel{\text{def}}{=}(p-1)\cdot p^d\cdot p^{-\beta}$. For an adequate constant $c'<0.1$, we shall pick $\beta=d+1-(c'/p\ln p)$. Then, $q=(p-1)\cdot p^{-1+(c'/p\ln p)}\approx\frac{p-1}{p}\cdot\frac{p+c'}{p}$ (or rather we pick c' such that $q=\frac{p-1}{p}\cdot\frac{p+0.1}{p}$). Hence, $q<\frac{p^2-(1-c')p}{p^2}<1-(0.9/p)$, assuming $c'<0.1$. It follows that $\sum_{i\geq1}q^{i-1}<\frac{1}{0.9/p}$ and $Q<Q'+Q''$, where $Q'\stackrel{\text{def}}{=}\frac{2.5^d}{p^\beta}\cdot p/0.9$ and $Q''\stackrel{\text{def}}{=}2.5^d\cdot(1-(0.9/p))^{k'}$. For large enough $c''>0$, setting $k'=c''\cdot pd$, we get $Q''=2.5^d\cdot(1-(0.9/p))^{c''pd}$, which is approximately $\exp(-(0.9c''-\ln 2.5)\cdot d)$. Hence, we can make $Q''>0$ arbitrary small by picking a large enough c''. As for Q', using $p^\beta=(p-1)\cdot p^d/q$ and $q=\frac{p-1}{p}\cdot\frac{p+0.1}{p}$, we have

$$\begin{aligned}Q' &= \frac{2.5^d\cdot(p/0.9)}{(p-1)p^d/q}\\ &= \frac{2.5^d\cdot(p/0.9)\cdot(p-1)(p+0.1)/p^2}{(p-1)p^d}\\ &= (2.5/p)^d\cdot\left(\frac{1}{0.9}+\frac{1}{9p}\right)\\ &\leq (2.5/3)^d\cdot\left(\frac{1}{0.9}+\frac{1}{27}\right)\end{aligned}$$

where the last inequality holds for $p\geq3$. The claim (i.e., $Q'+Q''<1$) follows since $(2.5/3)^d\cdot\frac{31}{27}<1$ for every $d\geq1$. ■

Combining Facts 5.6.1 and 5.6.2, the claim follows. ■

Conclusion. As stated above, Claim 5.6 implies the claims of Theorem 1.9, except that this was shown only in the case of $t=n$. We first observe that what Claim 5.6 actually shows is that, for some $\gamma<d$ and any t, it holds that $(1/t)\cdot\sum_{j\in[t]}|D_j|<t^\gamma$. In case $t<n$, this is actually stronger

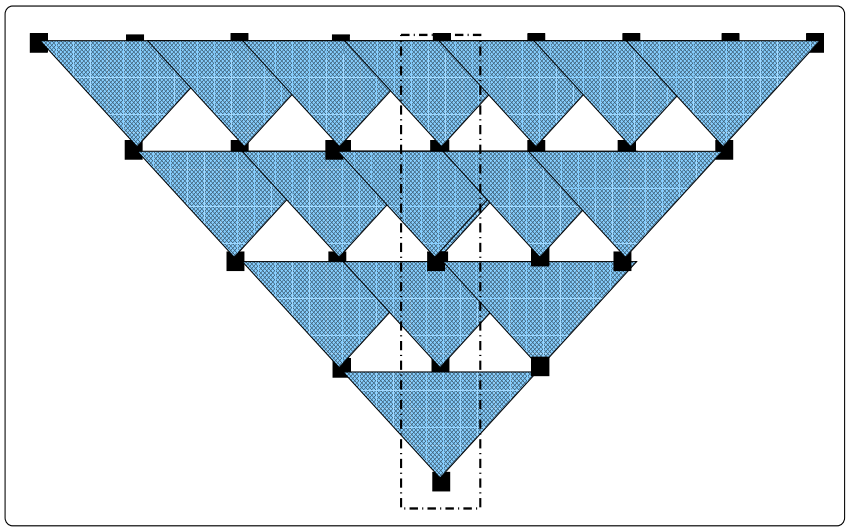


Figure 8: *A one-dimensional projection of the case of $t = 4p^\ell$ (for the analysis of the case of $t > n$). Only the shaded triangles (each of height p^ℓ) contain locations that may influence the value of the bottom location. The dashed rectangle marks the boundaries of $[t] \times [-(n-1), (n-1)]^d$.*

than what we need (and indeed in this case the tester has (total) time complexity $\mathrm{poly}(\epsilon^{-1}) \cdot t^\gamma$). But for $t > n$, we need a slightly different analysis.

Our first observation is that we should only care of location that are at distance at most $n-1$ from the origin (in max-norm), since only these location contain non-dummy values. That is, it suffices to upper-bound $(1/t) \cdot \sum_{j \in [t]} |D_j \cap [-(n-1), (n-1)]^d|$. Letting $\ell = \lceil \log_p 2n \rceil$, the second observation is that, for $j \in \mathbb{N}$, the set $D_j \cap [-(n-1), (n-1)]^d$ is a subset of $D_{j \bmod p^\ell}$, since $p^\ell \geq 2n$ and $D_{p^\ell} = \{-p^\ell, 0, p^\ell\}$ (see Figure 8).[26] Thus, for every $t \geq n$, it holds that

$$\begin{aligned} \frac{1}{t} \cdot \sum_{j \in [t]} |D_j \cap [-(n-1), (n+1)]^d| &\leq \frac{1}{t} \cdot \sum_{j \in [t]} |D_{j \bmod p^\ell}| \\ &\leq \frac{1}{t} \cdot \lceil t/p^\ell \rceil \cdot (p^\ell)^{1+\gamma} \\ &< 2 \cdot (2pn)^\gamma \end{aligned}$$

where the last inequality uses $p^\ell < p \cdot 2n$. Using $\gamma < d$, we conclude that for some $\gamma' < d$ it holds that $(1/t) \cdot \sum_{j \in [t]} |D_j \cap [-(n-1), (n-1)]^d| < n^{\gamma'}$. Hence, we get

Theorem 5.7 (Theorem 1.9, restated): *For any $d \geq 1$ and any field Σ of prime order there exists a constant $\gamma < d$ such that the following holds. For any linear $\Gamma : \Sigma^{3^d} \to \Sigma$ there exists a time-conforming oracle machine of* (total) *time complexity* $\mathrm{poly}(\epsilon^{-1}) \cdot \min(n, t)^\gamma$ *that tests the consistency of evolving environment with respect to $\Gamma : \Sigma^{3^d} \to \Sigma$ and fully visible states. Furthermore, the tester is nonadaptive and has one-sided error probability.*

[26] In other works, for every $i \in [p^\ell]$, it holds that $D_{p^\ell + i} \cap [-(n-1), (n-1)]^d$ is contained in $(D_{p^\ell} + D_i) \cap [-(n-1), (n-1)]^d$, which in turn equals D_i, since $p^\ell \geq 2n$.

We have obtained a sublinear complexity tester for linear rules over any finite field of prime order, which may be viewed as modular linear rules with a prime modulus. By using the Chinese remainder theorem, we can obtain a similar tester for modular linear rules with any modulus that is a product of *distinct* primes, but this result does not extend to general composite numbers. For starters:

Open Problem 5.8 *Can Theorem 5.7 be extended to arbitrary finite fields? More generally, we ask whether a tester with sublinear complexity exists for linear rules over any commutative ring.*

A more important open problem refers to the complexity of our testers. Recall that our testers have complexity that is only mildly sublinear. Even in the special case of Claim 5.2, the complexity is $n^{0.8}$ for an environment of size n. Can one obtain significantly better complexity? Specifically:

Open Problem 5.9 (Can Theorem 5.7 be drastically improved?) *For which $d \geq 1$ and $\gamma > 0$ does the following hold. For any field Σ of prime order, and any linear $\Gamma : \Sigma^{3^d} \to \Sigma$, there exists a* (time-conforming) *tester of* (total) *time complexity* $\mathrm{poly}(\epsilon^{-1}) \cdot \min(n,t)^{\gamma}$ *for the evolution rule $\Gamma : \Sigma^{3^d} \to \Sigma$* (and fully visible states)*. Getting even bolder, we ask whether polylogarithmic complexity is possible.*

6 Environments of Moving Objects: The Dense Case

In this section we consider dynamic environments that represent objects that are moving in a d-dimensional space.[27] A very simple environment of moving object consists of one in which, starting from their initial position, various objects move in fixed speed in some fixed direction. Such (d-dimensional) evolving environments can be modeled by (d-dimensional) cellular automata in which the states encode the presence of objects in the location as well as the direction in which they are moving. The simplest such model allows several objects to be present in the same location at the same time (but not at the initial time).

We observe that learning such d-dimensional environments requires $\Omega(n^d)$ queries, whereas testing is possible by $\mathrm{poly}(2^d/\epsilon)$ queries (and $\mathrm{poly}(2^d/\epsilon)$-time computations). Essentially, the tester consists of querying $O(1/\epsilon)$ random locations in $\mathtt{ENV}_1$, and querying $\mathrm{poly}(2^d/\epsilon)$ random locations in $\mathtt{ENV}$ that correspond to possible movements starting from each of these initial locations. That is, we uniformly select a set S of $O(1/\epsilon)$ locations in $[n]^d$ and a set T of $O(1/\epsilon)$ indices in $\{2,...,t\}$. Next, for each location $(i_1,...,i_d) \in S$ queried in $\mathtt{ENV}_1$, each possible direction $\overline{\delta} = (\delta_1,...,\delta_d) \in \{-1,0,1\}^d$, and each $j \in T$, we query $\mathtt{ENV}_j$ at $(i_1 + (j-1)\delta_1, ..., i_d + (j-1)\delta_d)$. For each such $(i_1,...,i_d)$ and $\overline{\delta}$, either for each $j \in \{1\} \cup T$ the value $\mathtt{ENV}_j(i_1 + (j-1)\delta_1, ..., i_d + (j-1)\delta_d)$ should indicate the presence of an object moving in direction $\overline{\delta}$ or none of these values should indicate it. Note that if the tester sees no violation of the foregoing condition (with sufficiently high constant probability), then $\mathtt{ENV}$ must be ϵ-close to a legal evolving environment (because all but at most $\epsilon/2$ of the possible movements starting from some $(i_1,...,i_d) \in [n]^d$ and proceeding in direction $\overline{\delta} \in \{-1,0,1\}^d$ are $\epsilon/2$-close to being consistent, or else our sample would have caught such an inconsistency). Hence:

Theorem 6.1 (testing uninterrupted moving objects, very loosely stated): *Let $\Gamma : \Sigma^{3^d} \to \Sigma$ be a local rule that captures the uninterrupted fixed-speed movement of objects in a d-dimensional grid.*

[27] In fact, the models described in this section are related to Chazelle's original interest in tracing the movement of objects in an environment (see Acknowledgments).

Then, there exists a time-conforming oracle machine of (total) *time complexity* $\mathrm{poly}(2^d/\epsilon)$ *that tests the consistency of evolving environments with respect to* $\Gamma : \Sigma^{3^d} \to \Sigma$ *and fully visible states.*

In Theorem 6.1 we considered very simple objects that move at the same fixed speed in one of a few directions and may cross each other's way without causing any interruption. In the current section, we consider somewhat more involved movements. Specifically, we consider objects that move at the same fixed speed in one of a few directions, as long as their paths do not cross; when their paths do cross, the objects just stop at their current location (and remain there forever). The analysis of this evolution rule turns out to be quite a bit more complicated, due to the interaction between the various moving objects. Confining ourselves to the one-dimensional case, we prove that consistency with respect to this evolution rule can also be tested by using $\mathrm{poly}(1/\epsilon)$ queries: See Theorem 1.10, which is proved in Section 6.1.

A more general (and complex) model refers to the case of objects that change their direction of movement (in a multi-dimensional environment) according to their internal state. In Section 6.2 we show that, in general, testing consistency of such d-dimensional moving objects is not easier than testing consistency of the evolution of d-dimensional environments with fully visible states. This is the case because such (stateful) moving objects can emulate the evolution of any environment having fully visible states.

For the sake of simplicity, we present the model in intuitive terms, rather than via a cellular automaton. Nevertheless, such a modeling can be provided by using a "semaphoring" mechanism (to avoid collision); for details, see Appendix A.2.

The model studied in the current section is most adequate for the case that the number of objects is comparable to the size of the environment; that is, the density of the objects in the environment is noticeable with respect to the proximity parameter (i.e., there are more than ϵn^d objects). An alternative model that is aimed at environment with very few objects is presented in Section 7.

6.1 A special case: Fixed one-dimensional interruptible movement

We consider the case of one-dimensional environments of moving objects that move in a fixed direction (until they stop since they collide with some other object). We encode such objects by the direction of their movement, represented by $-1, 0, +1$, which is visible (to the observer). A vacant location is represented by $\perp$. An object moves to a vacant location (in the direction of its movement) if no other object wishes to move to that location; otherwise it stops in its current location (i.e., remains in its current location (forever)). The latter case consists of two subcases: (1) more than one object wishes to move to the same location, and (2) the location that the object wishes to move to is occupied by an object that wishes to either stay there or move in the opposite direction. We also postulate that in the initial configuration the moving objects are not adjacent to one another.[28] This restriction allows to avoid the case that an object wishes to enter a location that is currently occupied by an object that wishes to move in the same direction.[29] The justification for this restriction is that in "real life" movement is continuous and such a problem

[28] A cellular automaton formulation of this model may assert that when being in a state that belongs to the initial states, the object is eliminated if it neighbors any other object.

[29] Note that this case is problematic because we may have a long sequence of such objects in which the first one (i.e., the one "in front") wishes to move into a location that is occupied by a standing object, in which case the movement of the entire sequence is postponed (but this case cannot be detected locally at the other end of the sequence).

will not arise. Since we work with a discretized space, we can simply select the discretization based on the minimum distance between objects.

(We note that a single evolution step of this environment can be emulated by a pair of steps of the cellular automaton described in Appendix A.2.)[30]

6.1.1 A two-sided error tester

It will be more convenient to associate the initial time period with 0 rather than with 1. Thus, we consider testing the evolution of environments of the form $\texttt{ENV} : \{0, 1, ..., t\} \times [n] \to \{-1, 0, +1, \perp\}$, where $\perp$ encodes an empty position and $\delta \in \{-1, 0, 1\}$ encodes an object that moves in direction δ (where $\delta = 0$ means that the object remains in place). The reader may think of $t = n$, but the analysis holds for any $t \in [\epsilon n, n]$. (The few places where we rely on these bounds will be indicated.) The other cases (i.e., $t < \epsilon n$ and $t > n$) will be treated separately at the end of this subsection.

In the following description we also refer to queries made to locations outside of the environment's domain (i.e., $[n]$); such queries are of course not made, and the tester never rejects based on them (i.e., at each check, the answer is fictitiously defined as one that will not cause rejection).

The tester. Given proximity parameter ϵ, the tester selects two random sets $S \subset [n]$ and $R \subset [t]$ of $m = \text{poly}(1/\epsilon)$ elements each, and augments R with 0 (i.e., $0 \in R$ always holds). It then conducts the following checks:

1. *Spaced initial configuration check*: The test checks whether the initial configuration contains no adjacent moving objects. That is, for every $s \in S$, the test queries $\texttt{ENV}_0(s)$ and $\texttt{ENV}_0(s+1)$, and checks that at least one of these two values is in $\{0, \perp\}$.

2. *Individual movement check*: The test checks whether the movement of individual objects (as well as their standing) is continuous; that is, if an object moved at time j, then it must have moved at every time prior to time j, whereas if it stayed in place in time j then it must stay in place at any time after j (i.e., if $\texttt{ENV}_j(i) = \delta \in \{\pm 1\}$, then $\texttt{ENV}_{j-1}(i - \delta) = \delta$, whereas if $\texttt{ENV}_j(i) = 0$, then $\texttt{ENV}_{j+1}(i) = 0$). This is checked in a rather straightforward manner, as explained next.

   Let $R = \{r_0, r_1, ..., r_m\}$ such that $0 = r_0 < r_1 < r_2 < \cdots < r_m$. Then, for every $s \in S$ and every $\delta \in \{\pm 1\}$, the test queries for the values of $\texttt{ENV}_0(s), \texttt{ENV}_{r_1}(s+\delta \cdot r_1), ..., \texttt{ENV}_{r_m}(s+\delta \cdot r_m)$, and checks that the sequence of answers is in $\delta^*\{0, -\delta, \perp\}^*$. Likewise, for every $s \in S$, the test queries for the values of $\texttt{ENV}_0(s), \texttt{ENV}_{r_1}(s), ..., \texttt{ENV}_{r_m}(s)$, and checks that the sequence of answers is in $\{\pm 1, \perp\}^*0^*$. For an illustration, see Figure 9.

3. *Matching movement and standing check*: While Step 2 checks that we have continuous movement and standing of objects, it does not check that the "standing" (of an object) starts when

[30] Following is a sketchy description of the emulation, which proceeds as follows: In the first sub-step, objects that wish to move are replaced by an "out-shadow" in the old location and an "in-shadow" appears in the new (vacant) location (provided that only one object wishes to enter the vacant location, otherwise the new location is marked as denying access). In the second sub-step, the out-shadow is modified according to the contents of the foregoing new location; in particular, if an in-shadow was marked, then it is transformed into the current residence of the object and the corresponding out-shadow is removed. In other words, the in-shadow is used as a semaphore in this primitive access control mechanism. Objects wishing to move to an occupied location are handled more easily (since they can be made to stop immediately, without resorting to any access control mechanism).

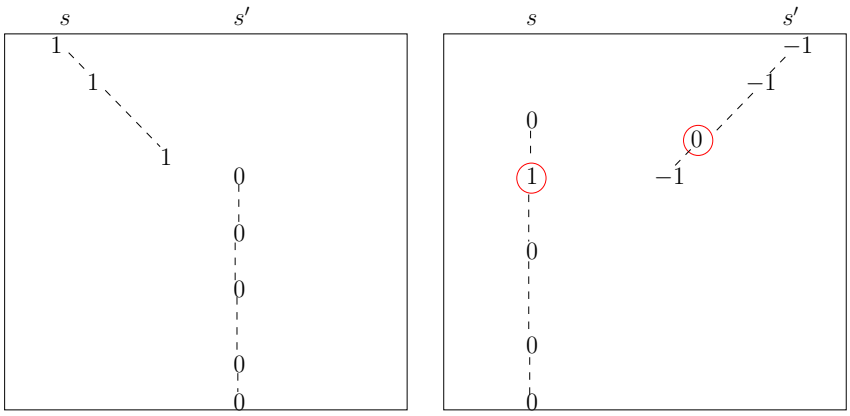


Figure 9: *An illustration for Step 2. The left image shows a legal configuration, whereas the right image shows an illegal one. The broken lines indicate the trajectories of the objects, and the violations are circled on the right.*

its movement ends (rather than have objects disappear or be created). To check this feature, the tester checks that the statistics regarding the ending of movements matches the statistics of standing objects. Details follow.

We say that an object with initial position $s \in S$ stopped approximately at time $r \in R$ if there exists $\delta \in \{\pm 1\}$ such that $\texttt{ENV}_r(s + \delta \cdot r) \neq \delta$ and for every $r' < r$ in R it holds that $\texttt{ENV}_{r'}(s + \delta \cdot r') = \delta$; in this case, the approximate stopping position of this object is defined as $s + \delta \cdot r$. Likewise, we say that an object started standing in position i approximately at time $r \in R$ if $\texttt{ENV}_r(i) = 0$ and for every $r' < r$ in R it holds that $\texttt{ENV}_{r'}(i) \neq 0$. In particular, objects that stand at time 0 are not considered as starting to stand (at time 0), and will not be included in this check. Ditto for moving objects that never stop. For an illustration, see Figure 10.

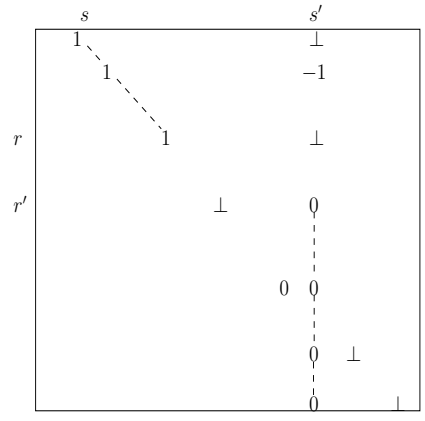


Figure 10: *An illustration for the notions of approximate stopping time and approximate time of starting to stand. The object that was in position s at time 0, stopped approximately at time r, and the object who is standing in position s' at time t, started to stand approximately at time r'.*

By using the queries made in Step 2, the tester compiles a list of pairs (r, i) such that $i \in S \pm R$

($\stackrel{\text{def}}{=} \{s_i \pm r_j : s_i \in S, r_j \in R\}$) is the approximate stopping position of an object that stopped at approximate time $r \in R$, and a list of pairs (r, i) such that some object started standing in position $i \in S$ approximately at time $r \in R$. For a fixed polynomial p, the tester checks if there exists a matching between the two lists such that the sum of the pairwise distances is smaller than $p(\epsilon) \cdot n|S|$, where an unmatched list element is charged $2n$ units and a matching of element (r_1, i_1) to element (r_2, i_2) is charged $|r_1 - r_2| + |i_1 - i_2|$ units. Details regarding the implementation of this check appear in the analysis of this step.

4. *Non-crossing movement check*: While Steps 2 and 3 refer to the individual movement and standing of objects, the current step refers to their pairwise interaction. Specifically, referring to the queries made in Step 2, the current step checks that no two objects have crossed each other's way. This is checked in a rather straightforward manner, as described next.

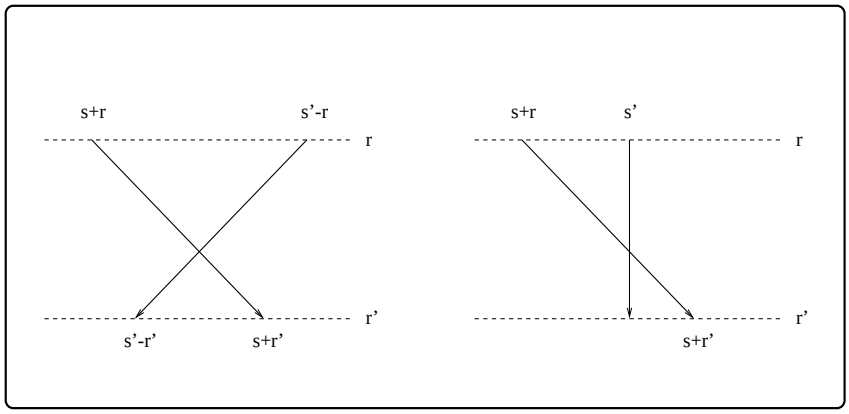


Figure 11: *An illustration for Step 4. The left image shows a crossing of two moving objects, whereas the right image shows an object moving to the right while crossing a standing object (at position s').*

For $k = 0, 1, ..., m-1$, the test rejects if there exist $s_1, s_2 \in S$ such that $\texttt{ENV}_{r_k}(s_1 + r_k) = \texttt{ENV}_{r_{k+1}}(s_1 + r_{k+1}) = 1$ and $\texttt{ENV}_{r_k}(s_2 - r_k) = \texttt{ENV}_{r_{k+1}}(s_2 - r_{k+1}) = -1$, whereas $s_1 + r_k < s_2 - r_k$ but $s_1 + r_{k+1} > s_2 - r_{k+1}$. This means that at time r_k, location $s_1 + r_k$ contains an object moving to the right and location $s_2 - r_k$, which is on the right of location $i_1 + r_k < s_2 - r_k$, contained an object moving to the left, but at time r_{k+1} the first object was positioned to the right of the second object: See the left image in Figure 11. Likewise, for $k = 0, 1, ..., m-1$, the test rejects if there exist $s_1, s_2 \in S$ such that $\texttt{ENV}_{r_k}(s_1 + r_k) = \texttt{ENV}_{r_{k+1}}(s_1 + r_{k+1}) = 1$ (resp., $\texttt{ENV}_{r_k}(s_1 - r_k) = \texttt{ENV}_{r_{k+1}}(s_1 - r_{k+1}) = -1$) and $\texttt{ENV}_{r_k}(s_2) = \texttt{ENV}_{r_{k+1}}(s_2) = 0$, whereas $s_1 + r_k < s_2$ but $s_1 + r_{k+1} > s_2$ (resp., $s_1 - r_k > s_2$ but $s_1 - r_{k+1} < s_2$): See the right image in Figure 11.

5. *Non-spaced standing check*: The purpose of this check is to test that whenever objects stop, they do so for a good reason (i.e., the location that they want to move into is occupied already). Thus, when an object moving in a certain direction stops (e.g., starts at position s and stops at time r at position $s + r$), all objects that move in the same direction and pass through the initial position of the first object (i.e., s) must stop in subsequent locations (i.e., without leaving gaps among them). See illustration in Figure 12.

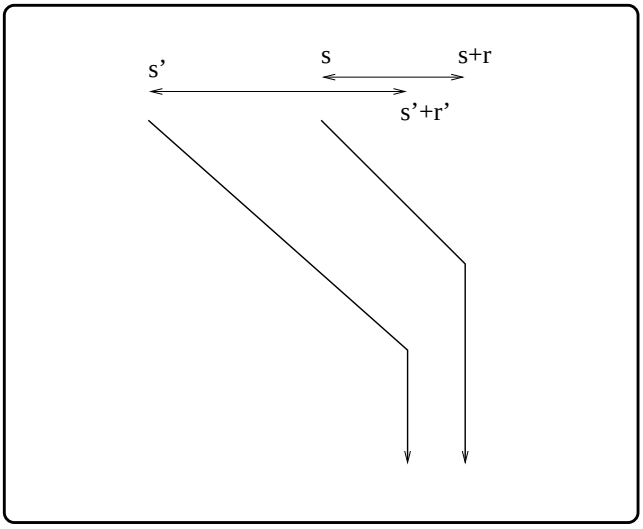


Figure 12: *An illustration for Step 5. The first object moves from position s to position $s+r$, where it stops, whereas the second objects moves from $s' < s$ to $s'+r' \in [s, s+r)$. In this case, the interval $[s'+r', s+r]$ must be filled with stopping objects (at time t).*

This check is performed as follows. The tester considers the intervals of movements that it has seen in prior steps. For each $s \in S$ such that $\texttt{ENV}_0(s) \in \{+1, -1\}$, let $r^+(s)$ denote the approximate stopping time of the object with initial position s (as defined in Step 3) and let $r^-(s) \stackrel{\text{def}}{=} \max\{r \in R : r < r^+(s)\}$. By the definition of the approximate stopping time of on object, the exact time that the object with initial position s is supposed to stop is in the interval $(r^-(s), r^+(s)]$; that is, it is still moving at time $r^-(s)$ and is no longer moving at time $r^+(s)$. For any pair of objects that both move to the right (resp. to the left) and have initial starting positions s and $s' < s$ (resp. $s' > s$), if $s'+r^-(s')+1 \geq s$ (resp. $s'-r^-(s')-1 \leq s$) the test considers the interval $I = [s'+r^+(s'), s+r^-(s)+1]$ (resp. $I = [s-r^-(s)-1, s'-r^+(s')]$). If there exists a point $p \in S \cap I$ such that $\texttt{ENV}_t(p) \neq 0$, then the tester rejects. For an illustration, see Figure 13.

Loosely speaking, except for Step 3, the tester essentially checks whether the values obtained from ENV are consistent with a legal evolution of the environment (from some legal initial configuration). Step 3 goes beyond such a consistency requirement: It also asks whether the statistics of stopping and starting-to-stand times match. Indeed, such a check, which relies on statistics, has two-sided error. As will be shown in Theorem 6.8, two-sided error is essential for any tester (for this rule) that has complexity that does not depend on the size of the environment (i.e., n).

Clearly, any environment ENV that is consistent with the ("moving object") evolution rule is accepted with very high probability. The (small) error probability is due to Step 3, which performs an estimation based on a random sample.

The rest of the analysis is devoted to showing that if the tester accepts with high probability (say, with probability at least 2/3), then ENV is ϵ-close to an evolution that is determined by the foregoing rule of fixed-speed movement with stopping. We proceed in a sequence of steps, where in each step we rely on the fact that a specific check of the tester passed in order to show that the currently considered environment is close to one that satisfies yet another constraint.

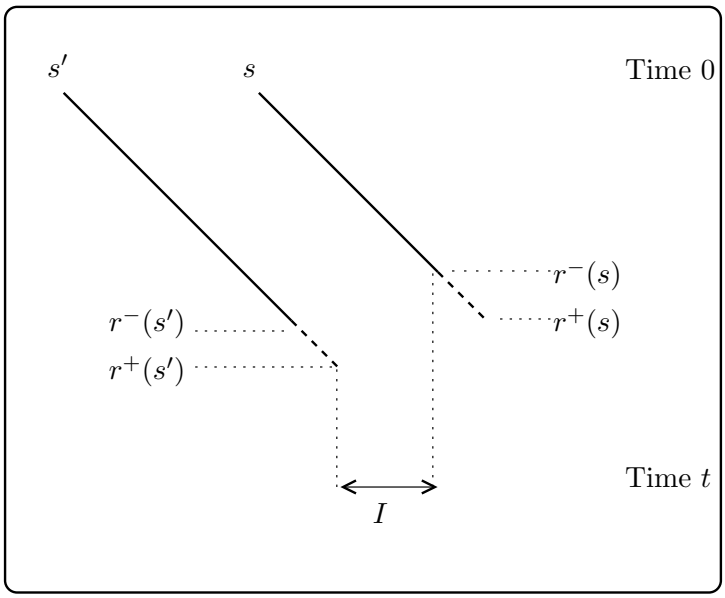


Figure 13: *An illustration for the check performed in Step 5. The solid segments indicate that the object is moving, and the dashed segments indicate that the object is either moving or already stopped. The interval I must fully consist of standing objects at time t.*

It will be convenient to associate a separate random sample to each step; that is, Step i uses a sample $S_i \subset S$ and $R_i \subset R$. Note that the fact that the actual steps use the same (bigger) sample only improves their quality. This is obvious for the checks that have a one-sided error probability, but also holds for Step 3.

Inferring from the success of Steps 1 and 2. Looking at $\texttt{ENV}$, for every $i \in [n]$ and $\delta \in \{\pm 1\}$, we consider a possible moving object that starts in location i (at time 0) and moves in direction δ. We call such a movement legal if the sequence of values $\texttt{ENV}_0(i), \texttt{ENV}_1(i+\delta), \ldots, \texttt{ENV}_t(i+\delta \cdot t)$ is in $\delta^*\{0, -\delta, \perp\}^*$. We say that the sequence is γ-close to being legal for $\gamma \in [0,1]$ if it can be made legal by modifying at most γt values in the sequence.

If Step 2 accepts (with probability at least 2/3), then it must be that, for all but at most $\text{poly}(\epsilon) \cdot n$ of the pairs $(i, \delta) \in [n] \times \{\pm 1\}$, the movement that starts at $i \in [n]$ in direction $\delta \in \{\pm 1\}$ is $\text{poly}(\epsilon)$-close to being legal. Now, we omit the few exceptions, and correct the rest of the movements so that they are legal. We address the issue of different corrections of the same cell (due to collisions of objects) subsequently.

For each such moving object we call its actual movements (i.e., the δ^*-prefix of $\texttt{ENV}_0(i), \texttt{ENV}_1(i+\delta), ..., \texttt{ENV}_n(i+\delta n)$) a diagonal line. Likewise, we can correct the vertical lines so that they become legal, where the vertical line at position i is legal if the sequence of values $\texttt{ENV}_0(i), \texttt{ENV}_1(i)..., \texttt{ENV}_n(i)$ is in $\{\pm 1, \perp\}^* 0^*$. In the sequel, we refer to its 0^*-suffix as a vertical line.

Finally, relying on Step 1, we can omit the few (non-empty) diagonal lines that are adjacent to other (non-empty) diagonal lines. Thus, we obtain an environment $\texttt{ENV}'$ that is ϵ_1-close to $\texttt{ENV}$, for $\epsilon_1 = \text{poly}(\epsilon)$ to be determined later. Indeed, in $\texttt{ENV}'$ the initial configuration (i.e., $\texttt{ENV}'_0$) contains no adjacent moving objects, and the movement of individual objects in $\texttt{ENV}'$ is continuous; that is, $\texttt{ENV}'$ consists of a collection of diagonal lines and vertical lines (but there is no guarantee as to the

relation between these lines). Note, however, that $\texttt{ENV}'$ may contain illegal symbols for the (possibly created) collisions of two objects in same cell; this illegality will be removed in the analysis of Step 4 (which is handled after handling Step 3).

Inferring from the success of Step 3. The analysis consists of two sub-steps: First we use the hypothesis that the check passes (with high probability) in order to argue that the statistics of the actual stopping and standing pairs in $\texttt{ENV}'$ (rather than the approximate locations viewed in the sample of ENV) are close. Once this is done, we shall close the gap (between the stopping and starting locations) in a way that corresponds to legal movements of objects. Note that the foregoing claim is made with respect to $\texttt{ENV}'$, whereas the tester tests ENV. However, our analysis will refer to a number of samples m' that is sufficiently small such that $m'\epsilon_1$ is very small (and so this analysis (which corresponds to a part of the tester) cannot tell the difference between $\texttt{ENV}'$ and ENV). The first step in the analysis corresponds to the following problem, which is of independent interest, where in our case $d = 2$ and we are interested in the ℓ_1-norm.

Definition 6.2 (the matching distance between sets of points in $[0,1]^d$): *Let $P = \{p_1, ..., p_n\}$ and $Q = \{q_1, ..., q_n\}$ be sets of points in $[0,1]^d$, and define the pairwise distance between these sets as*

$$\Delta(P,Q) = \min_{\pi \in \mathrm{Sym}(n)} \left\{ \sum_{i \in [n]} \|p_i - q_{\pi(i)}\| \right\} \tag{11}$$

where $\mathrm{Sym}(n)$ *denotes the set of permutations over $[n]$ and $\|\cdot\|$ denotes some fixed norm over $\mathbb{R}^d$.*

The problem is to approximate $\Delta(P,Q)$, when obtaining samples from P and from Q. We assume, without loss of generality, that $\|r\| \leq d$ for every $r \in [0,1]^d$.

Claim 6.3 (estimating the matching distance between sets of points in $[0,1]^d$): *The pairwise distance between two n-sets can be approximated up to an additive deviation of $\epsilon' n$ by a probabilistic $\mathrm{poly}(1/\epsilon')$-time algorithm that can obtain samples from both sets. Furthermore, the algorithm outputs the value of $(n/m) \cdot \Delta(P'', Q'')$, where P'' and Q'' are sets of $m = \mathrm{poly}(1/\epsilon')$ points selected uniformly and independently in the corresponding n-sets.*

We comment that, for $d = 1$, an optimal permutation π is obtained by sorting both sets and matching the i^{th} element of P to the i^{th} element of Q (see Remark 6.4).

Proof: For $\varepsilon = \epsilon'/10d$, consider a discretization of all points such that each point reside in $\{((i_1 - 0.5) \cdot \varepsilon, ..., (i_d - 0.5) \cdot \varepsilon) : i_1, ..., i_d \in [1/\varepsilon]\}$ and is at distance (i.e., $\|\cdot\|$-distance) at most $d\varepsilon/2$ from its original location. Denote the resulting multi-sets by P' and Q', respectively. Clearly, $\Delta(P', Q') = \Delta(P, Q) \pm d\varepsilon n$.

Next, consider taking m-sized samples from P' and Q', denoted P'' and Q'', and consider the multi-sets P''' and Q''' obtained by repeating each element in the sample n/m times. Since, with very high probability, the element-counts in the multi-sets P''' and Q''' are very similar to the counts in P' and Q', it holds that $\Delta(P''', Q''') = \Delta(P', Q') \pm \varepsilon n$. Finally, observe that $\Delta(P''', Q''') = \frac{n}{m} \cdot \Delta(P'', Q'')$, where the lower bound holds by observing that the permutation π''' used in $\Delta(P''', Q''')$

yields a permutation π'' for $\Delta(P'', Q'')$.[31] We note that $\Delta(P'', Q'')$ can be computed in poly(m)-time by finding a perfect matching of minimum weight in the bipartite graph defined by the distances between points in P'' and points in Q''. ■

Remark 6.4 (computing the matching distance between sets of points in $[0,1]$, a detour): *In the case of $d = 1$, a permutation π that obtains the value of Eq. (11) can be found by sorting both sets and matching the i^{th} element of P to the i^{th} element of Q. The claim can be proved by considering $\{p_1, ..., p_m\} \subset \mathbb{R}$ and $\{q_1, ..., q_m\} \subset \mathbb{R}$ such that $p_1 < \cdots < p_m$ and $q_1 < \cdots < q_m$ and showing that $\sum_{i\in[n]} |p_i - q_i|$ equals $\min_\pi\{\sum_{i\in[n]} |p_i - q_{\pi(i)}|\}$. To show this, let π a permutation achieving the latter minimum, and let $i \in [n]$ be smallest such that $\pi(i) \neq i$. Letting $j = \pi(i)$ and $k = \pi^{-1}(i)$, we observe that $|p_i - q_i| + |p_k - q_j| \leq |p_i - q_j| + |p_k - q_i|$ and the claim follows (by considering a permutation π for which $\min(i : \pi(i) \neq i)$ is largest).*[32]

Turning back to the analysis of Step 3, let $\{(r'_i, s'_i) : i \in [n']\}$ denote the set of ending positions of diagonal lines (i.e., the stopping positions of moving objects), and $\{(r''_i, s''_i) : i \in [n'']\}$ denote the starting positions of vertical lines (i.e., positions in which objects started standing). Defining $p_i = (r'_i, s'_i)/2n$ (resp., $q_i = (r''_i, s''_i)/2n$) if $i \in [n']$ (resp., if $i \in [n'']$) and $p_i = (1, 1)$ (resp., $q_i = (1, 1)$) otherwise, we apply Claim 6.3. Note that since $t \leq n$, the original positions in $\{0, 1, ..., t\} \times [n]$ are mapped to positions in $[0, 0.5]^2$, and matching the image of such a position to a fictitious position (i.e., $(1, 1)$) carries a large cost (i.e., distance at least 0.5). Assuming that, with high probability, Step 3 did not reject, we infer that the matching distance between the p_i's and the q_i's is at most poly$(\epsilon) \cdot n$. (Note that Step 3 effectively samples these points, except that it uses good approximations for their locations rather than their actual values.)[33]

It follows that all but at most poly$(\epsilon) \cdot n$ of the (ending positions of the) diagonals and the (starting positions of the) verticals can be matched to one another such that the distance between each pair of matched points is at most poly$(\epsilon) \cdot n$. (This holds when testing $\texttt{ENV}$, although the claim refers to $\texttt{ENV}'$, because the two are ϵ_1-close and the size of the sample that we took is smaller than $1/10\epsilon_1$.)[34]

Omitting the exceptional lines (i.e., the lines left unmatched or pairs of lines that are matched at a large distance), we remain with small gaps in the remaining pairs of lines, where each gap is of size at most poly$(\epsilon) \cdot n$, which is small compared to $t \geq \epsilon n$. The generic cases for such gaps (up to rotation) are depicted in Figure 14. In all cases, the diagonal line and the vertical line are extended or truncated to the crossing position marked 'X'. Hence, we obtain an environment $\texttt{ENV}''$ that is ϵ_2-close to $\texttt{ENV}$, where $\epsilon_2 = \text{poly}(\epsilon) > \epsilon_1$ (since we used $m = \text{poly}(1/\epsilon_2) < 1/10\epsilon_1$). Indeed,

[31] Observe that π''' defines a n/m-regular bipartite graph with multiple edges crossing the bipartition (P'', Q''), where $|P''| = |Q''| = m$. Coloring the edges of this bipartite graph with n/m colors, it follows that there exists a color that corresponds to a perfect matching of minimum sum of distances. This matching yields the desired permutation π''.

[32] In proving this observation, we may assume, w.l.o.g, that $p_i < q_i$ (and note that $j, k > i$). If $p_k > q_i$, then $|p_i - q_j| + |p_k - q_i| \geq |[p_i, \max(p_k, q_j)]| > |[p_i, q_i]| + |p_k - q_j|$. If $p_k \leq q_i$, then $|p_i - q_j| + |p_k - q_i| = |[p_i, q_j]| + |[p_k, q_i]| = |[p_i, q_i]| + |[p_k, q_j]|$.

[33] Indeed, two points are being made here. The first is that by selecting a random $s \in [n]$ and considering the diagonal $(r, s + r)_{r\in[t]}$ (resp., $(r, s - r)_{r\in[t]}$), Step 3 samples the ending positions, where the cases of $\texttt{ENV}_0(s) \in \{0, \perp\}$ and of a diagonal that does not stop are handled as generating a fictitious point. The same holds with respect to sampling the starting positions of vertical lines. The second point is that Step 3 uses the approximate ending (and starting) positions rather than the actual ones, but the approximation is good enough.

[34] Hence, the sample taken for the estimation hits a point on which $\texttt{ENV}$ and $\texttt{ENV}'$ differ with probability at most $1/10$.

ENV″ retains the foregoing features of ENV′, and in addition the stopping places of its diagonal lines match the starting places of vertical lines. In other words, ENV″ consists of lines that go from the first row to the last row such that each line consists of an initial (possibly empty) diagonal segment followed by a (possibly empty) vertical segment. (But these lines may cross one another.)

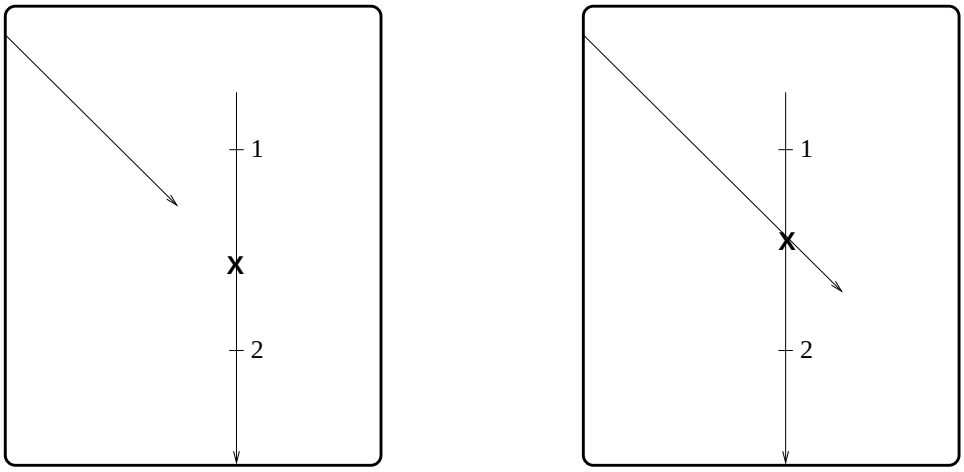


Figure 14: *Analysis of Step 3. The left image shows (part of) a diagonal line that stops too early (with respect to its matched vertical line, which starts at either location 1 or location 2), whereas the right image shows a diagonal that stops too late.*

Inferring from the success of Step 4. Considering the environment ENV″, we note that there may be line-crossings of two types: (1) crossing between lines that move in the same direction (with one turning vertical and crossing the other that continues as diagonal), and (2) crossing between lines that move in opposite directions (or between diagonals and lines that are vertical all along). We deal with each of these cases separately.

Starting with lines that describe a right movement (i.e., $\delta = +1$), we denote the start and end position of these lines by $(s_1, e_1), ..., (s_{n'}, e_{n'})$; that is, the i^{th} line starts at location s_i at time 0, and ends at location $e_i \geq s_i$ at time t. Let t_i denote the time in which the corresponding object stopped (i.e., the i^{th} line becomes vertical), so that $t_i = e_i - s_i$. We assume, w.l.o.g., that $s_1 < s_2 < \cdots < s_{n'}$, and let $\pi : [n'] \to [n']$ denote the (unique) permutation that "sorts" the end-locations; that is, $e_{\pi(1)} < e_{\pi(2)} < \cdots < e_{\pi(n')}$. We eliminate the crossing among these lines by modifying the lines such that the i^{th} line (which starts at location s_i) ends at location $e_{\pi(i)}$ rather than at location e_i, obtaining an environment ENV‴. The simple case in which a single crossing is eliminated (i.e., $\pi(i) = j$ and $\pi(j) = i$) is depicted in Figure 15; note that in this case the cost is $2 \cdot (s_j - s_i) + 2 \cdot \sqrt{2}(e_i - e_j)$, which is smaller than $3 \cdot (|s_i - s_j| + |e_i - e_j|)$.

In general, $\pi(i) = j$ may not imply $\pi(j) = i$, yet we claim that the cost of the entire correction is smaller than $2 \cdot \sum_{i \in [n']} (|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|)$. This is shown by charging each i (such that $\pi(i) \neq i$) one diagonal segment and one vertical segment. Each such segment is either removed or added, and the charging rule is described next. Recall that $t_i = e_i - s_i$ denotes the time in which the object corresponding to line i stops moving. Let $t'_i = e_{\pi(i)} - s_i$ denote the time in which the object corresponding to line i stops moving after we eliminate the crossings as described above. Observe that $t'_i - t_i = e_{\pi(i)} - e_i$ and that $t'_i - t_{\pi(i)} = s_i - s_{\pi(i)}$.

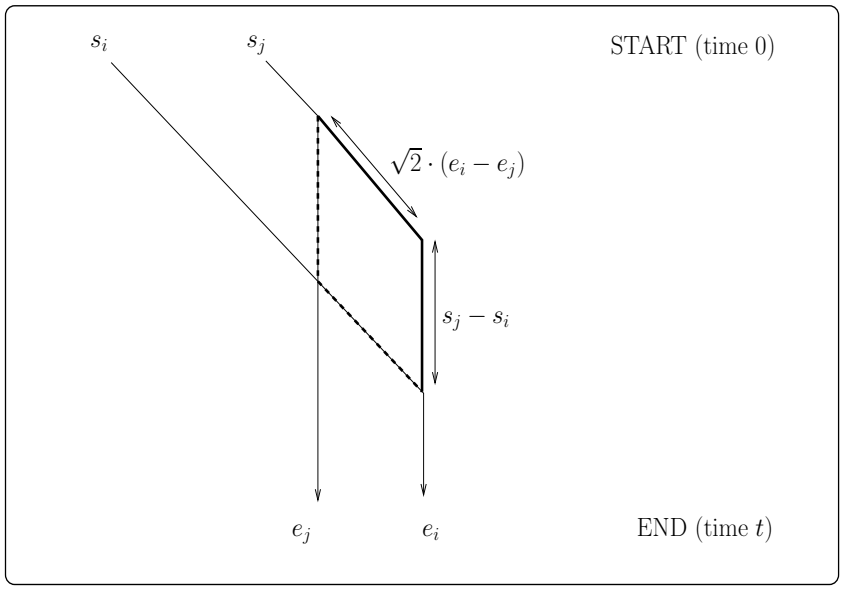


Figure 15: *Eliminating a crossing between lines i and j in the analysis of Step 4 for the case that $\pi(i) = j$ and $\pi(j) = i$. The segments that are omitted are shown in dashes, and the segments that are added are shown in bold.*

- Diagonal segment. We charge i with the diagonal segment between (t_i, e_i) and $(t'_i, e_{\pi(i)})$. If $e_i > e_{\pi(i)}$, implying that $t_i > t'_i$, then this segment is removed, and if $e_i < e_{\pi(i)}$, implying that $t_i < t'_i$, then this segment is added. The length of this segment is $\sqrt{(t_i - t'_i)^2 + (e_i - e_{\pi^{-1}(i)})^2}$, and since $|t_i - t'_i| = |e_i - e_{\pi^{-1}(i)}|$ this length equals $\sqrt{2} \cdot |e_i - e_{\pi^{-1}(i)}|$.
- Vertical segment. We charge i with the vertical segment between $(t'_i, e_{\pi(i)})$ and $(t_{\pi(i)}, e_{\pi(i)})$. If $i < \pi(i)$, implying that $s_i < s_{\pi(i)}$ and hence $t'_i > t_{\pi(i)}$, then this segment is removed, and if $i > \pi(i)$, implying that $s_i > s_{\pi(i)}$ and hence $t'_i < t_{\pi(i)}$, then this segment is added. The length of this segment is $|t'_i - t_{\pi(i)}| = |s_i - s_{\pi(i)}|$.

For an illustration of the charged segments, see Figure 16. The claim follows; that is, the cost of the entire correction is smaller than $2 \cdot \sum_{i \in [n']} (|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|)$.

On the other hand, the number of crossings (i.e., $|\{(i,j) \in [n']^2 : i < j \wedge e_i > e_j\}|$, which equals $|\{(i,j) \in [n']^2 : i < j \wedge \pi^{-1}(i) > \pi^{-1}(j)\}|$) is at least $\frac{1}{2} \sum_{i \in [n']} |i - \pi^{-1}(i)| = \frac{1}{2} \sum_{i \in [n']} |i - \pi(i)|$. To verify this observe that if $\pi^{-1}(i) > i$ (that is, e_i appears in position greater than i in the sorted order of the e_i's), then the number of lines j such that $j > i$ and $e_j < e_i$ must be at least $p^{-1}(i) - i$ and an analogous argument holds for the case that $\pi^{-1}(i) < i$ regarding lines j such that $j < i$ and $e_j > e_i$. The next claim implies that $\sum_{i \in [n']} |i - \pi(i)|$ can be lower bounded in terms of $\sum_{i \in [n']} (|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|)$; hence, if Step 4 rejects with small probability, then it must be the case that the correction cost is low.[35]

Claim 6.5 (distances versus ranking with respect to sorted order on the line): *Let $r_1 < r_2 < \cdots <$*

[35] Again, we use the fact that the size of the sample required to detect a crossing is small enough so that this sample cannot distinguish `ENV` from `ENV''`, which is ϵ_2-close to it.

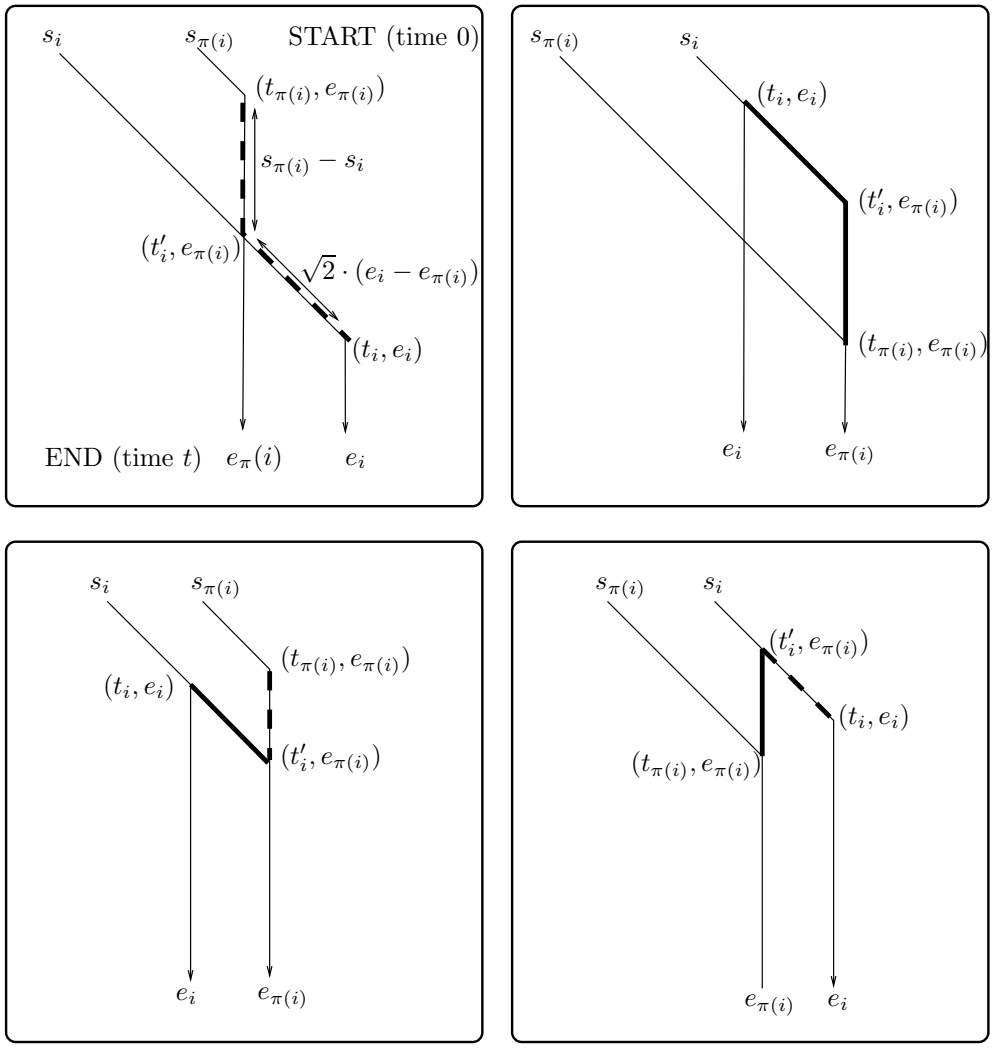


Figure 16: *An illustrations for the charging rule in the analysis of Step 4. The omitted segments are shown in dashes and the added segments are shown bold. In all cases line i is charged for either the omission or the addition of the diagonal segment whose endpoints are (t_i, e_i) and $(t'_i, e_{\pi(i)})$ and for either the omission or the addition of the vertical segment whose endpoints are $(t'_i, e_{\pi(i)})$ and $(t_{\pi(i)}, e_{\pi(i)})$. The four cases correspond to the four possibilities with respect to the relation between s_i and $s_{\pi(i)}$ and between e_i and $e_{\pi(i)}$. The lengths of the segments appear in the top left illustration.*

r_n be real numbers such that $r_n \leq r_1 + n$ and $\pi : [n] \to [n]$ be a permutation. If $\sum_{i\in[n]} |r_i - r_{\pi(i)}| > \varepsilon n^2$, then $\sum_{i\in[n]} |i - \pi(i)| > \mathrm{poly}(\varepsilon) \cdot n^2$.

Claim 6.5 extends to the case that we have only $n' < n$ points, by introducing $n - n'$ dummy points (such that $\pi(i) = i$ for $i > n'$). Applying Claim 6.5 twice (once with $s_1, ..., s_{n'}$ and π and once with $e_{\pi(1)}, ..., e_{\pi(n')}$ and π^{-1}), we infer that if the corrected environment $\texttt{ENV}'''$ (obtained from $\texttt{ENV}''$ as described above) is ϵ'-far from $\texttt{ENV}''$ (i.e., $2 \cdot \sum_{i\in[n']}(|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|) > \epsilon' t \geq \epsilon'' n^2$), then $|\{(i,j) \in [n']^2 : i < j \wedge \pi(i) > \pi(j)\}| > \mathrm{poly}(\epsilon'') \cdot n^2$. But in the latter case Step 4 will reject with

high probability (when inspecting either $\texttt{ENV}''$ or $\texttt{ENV}$), since most of the crossings do not occur close to the end of the line segments. The reason is simply that for each line and value v, the number of lines that can cross the line at distance at most v from the end of its diagonal segment (beginning of its vertical segment) is at most v. Thus, the hypothesis that the test accepts (with high probability) implies that $\texttt{ENV}'''$ is ϵ_3-close to $\texttt{ENV}$.

Proof: We partition $[n]$ into buckets, $B_{j,k}$ for $j,k \in [c]$, where $c = 2/\varepsilon$, such that $i \in B_{j,k}$ if $r_i \in [(j \pm 0.5)\varepsilon n/2)$ and $r_{\pi(i)} \in [(k \pm 0.5)\varepsilon n/2)$. The contribution of $\bigcup_{j\in[c]} B_{j,j}$ to $\sum_{i\in[n]} |r_i - r_{\pi(i)}|$ is at most $\varepsilon n^2/2$, since each $i \in B_{j,j}$ contributes at most $\varepsilon n/2$, and it follows that there exist $j \neq k$ such that $\sum_{i\in B_{j,k}} |r_i - r_{\pi(i)}| > \varepsilon n^2/2c^2 = \varepsilon^3 n^2/8$. Note that $|B_{j,k}| > \frac{\varepsilon^3 n^2/8}{n} = (\varepsilon/2)^3 \cdot n$, since $\max_i\{|r_i - r_{\pi(i)}|\} \leq n$. Let $B_{j,k} = \{i_1, ..., i_m\}$ where $i_1 < i_2 < \cdots < i_m$, and note that for every $i \in B_{j,k}$ it holds that $i \in [i_1, i_m]$ but $\pi(i) \notin [i_1, i_m]$, since $[r_{i_1}, r_{i_m}] \subseteq [(j \pm 0.5)\varepsilon n/2)$ whereas $r_{\pi(i)} \in [(k \pm 0.5)\varepsilon n/2)$ and $[(j \pm 0.5)\epsilon' n/2) \cap [(k \pm 0.5)\epsilon' n/2) = \emptyset$ for $j \neq k$. Let $B' = \{i \in B_{j,k} : \pi(i) < i_1\}$ and $B'' = \{i \in B_{j,k} : \pi(i) > i_m\}$. Then,

$$\begin{aligned}
\sum_{i\in B_{j,k}} |i - \pi(i)| &= \sum_{i\in B'} (i - \pi(i)) + \sum_{i\in B''} (\pi(i) - i) \\
&= \sum_{i\in B'} ((i_1 - \pi(i)) + (i - i_1)) + \sum_{i\in B''} ((i_m - i) + (\pi(i) - i_m)) \\
&= \sum_{i\in B'} |i_1 - \pi(i)| + \sum_{i\in B'} |i - i_1| + \sum_{i\in B''} |i_m - i| + \sum_{i\in B''} |\pi(i) - i_m| \\
&\geq 2 \sum_{i\in[|B'|]} i + 2 \sum_{i\in[|B''|]} i
\end{aligned}$$

where the inequality follows since each of the four sums is a sum of distinct positive integers. The claim follows since $|B'|^2 + |B''|^2 \geq 2 \cdot (m/2)^2$. ■

The foregoing description refers to crossings among the set of lines that move from left to right, but the same applies to the set of lines that go in the opposite direction. Hence, we eliminate all crossings among pairs of lines that go in the same direction (i.e., Type (1)). We now turn to crossings between lines that move in opposite directions and between diagonals and (full) verticals (i.e., Type (2)), where by verticals we refer to lines that are vertical from the start (i.e., from time 0). We shall actually first deal with this (sub)type of crossing, and before doing so we will convert lines that are almost vertical (i.e., which have a very short diagonal segment) into perfectly vertical lines.

Specifically, we call a line almost vertical if its diagonal segment is shorter than $\epsilon_3 \cdot n$. We turn all almost vertical lines to vertical at a relative cost of ϵ_3, while possibly introducing new crossings between verticals and diagonals. Abusing notation, let $\texttt{ENV}'''$ denote the resulting environment and note that $\texttt{ENV}'''$ is $2\epsilon_3$-close to $\texttt{ENV}$. We first relate the number of vertical-vs-diagonal crossings to the number of lines that participate in them. This is done by using the following claim (where points represent verticals and intervals represent diagonal segments).

Claim 6.6 (vertex cover versus number of edges in some interval graphs): *Let $p_1, \ldots, p_n \in [0,1]$ be points and $I_1, \ldots, I_n$ be intervals that are internal to $[0,1]$ such that $|I_j| > \epsilon'$ for every j. Consider the bipartite graph with vertex set $\{p_i : i \in [n]\} \cup \{I_j : j \in [n]\}$ such that (i,j) is an edge if and only if $p_i \in I_j$. Then, for every $\epsilon'' > 0$, the number of edges in this graph is at least $\epsilon'\epsilon'' \cdot n \cdot (\tau - \epsilon'' n)/2$, where τ is the size of a minimum vertex cover in this graph.*

Indeed, a vertex cover in this bipartite graph corresponds to a set of lines that when omitted from the current environment yields an environment in which there are no crossing between vertical lines and lines having diagonal segments. Claim 6.6 asserts that if this number must be big (i.e., bigger than $2\epsilon''n$), then there are many (i.e., $\text{poly}(\epsilon'\epsilon'') \cdot n^2$) pairwise crossings.

Proof: We show that if the edge density is low, then the graph has a small vertex cover. Specifically, we construct a vertex cover of the graph in iterations, where in each iteration we add a single vertex (i.e., an I_j) to the vertex cover and drop many edges from the current graph. When we complete the process only few vertices are left and so adding these to the vertex cover is fine. Details follow.

We consider a fixed partition of $[0,1]$ into $1/\epsilon'$ consecutive segments, denoted $S_1, \ldots, S_{1/\epsilon'}$, such that $S_i = [(i-1)\epsilon', i\epsilon')$ for $i < 1/\epsilon'$ and $S_i = [(i-1)\epsilon', i\epsilon')$ for $i = 1/\epsilon'$. At each iteration, we maintain in the current graph only non-isolated vertices. If there exists a segment S_k that contains at least $\epsilon''\epsilon'n$ point vertices (i.e., p_i's), then we consider the median point p_i in that segment; hence, $p_i \in S_k$ whereas $|\{j : p_j \in S_k \wedge p_j \leq p_i\}| \geq |S_k|/2$ and $|\{j : p_j \in S_k \wedge p_j \geq p_i\}| \geq |S_k|/2$. Let I_j be an interval that contains p_i (i.e., (i,j) is an edge in the residual graph). Then, I_j contains at least half of the points in S_k, since I_j (which has length greater than ϵ') covers p_i as well as one of the endpoints of S_k (and thus it covers all points in between). Thus, adding I_j to the vertex cover and omitting all edges that it covers, we increased the vertex cover by one unit while omitted at least $|S_k|/2 \geq \epsilon''\epsilon'n/2$ edges. The process stops when no segment contains $\epsilon''\epsilon'n$ point vertices that are non-isolated in the current graph, which means that the residual graph contains at most $e''n$ non-isolated point vertices. Thus, i iterations yields a vertex cover of size at most $i + \epsilon''n$, whereas the number of edges omitted in these i iterations is at least $i\epsilon''\epsilon'n/2$. Denoting the minimum vertex cover of the original graph by τ, we get $i \geq \tau - \epsilon''n$ and so the number of edges in the original graph is at least $(\tau - \epsilon''n) \cdot \epsilon''\epsilon'n/2$. The claim follows. ■

Hence, assuming that Step 4 rejected with very small probability, we infer that the number of pairwise crossings between (almost) verticals and diagonal segments is small (i.e., smaller than $\text{poly}(\epsilon) \cdot n^2$). Using Claim 6.6, it follows that few lines (i.e., $\text{poly}(\epsilon) \cdot n$) can be omitted from the environment such that all these crossing are eliminated. (Using $t \geq \text{poly}(\epsilon) \cdot n$, it follows that such an omission yields an environment that is $\text{poly}(\epsilon)$-close to $\texttt{ENV}'''$.)

A similar argument can be applied to crossing between diagonals that move in opposite directions. In this case we consider each of the two endpoints of diagonals that move in one direction against the intervals of movement of the diagonals that move in the other direction. (This is done twice, once per each direction playing the first role; see below.) Note that the number of edges in each application may be twice the number of crossing (since each line of the opposite direction contributes two endpoints), whereas a lack of edges between an interval and the endpoints of an (opposite direction) interval means that the former interval is internal to the latter (in which case edges will appear in the other application). Details follow.

Let $\{R_i\}_{i \in [n']}$ (resp., $\{L_i\}_{i \in [n'']}$) denote the set of intervals that correspond to diagonals that move to the right (resp., left). Now, consider one invocation of Claim 6.6 in which the R_i's play the role of the intervals and the endpoints of the L_i's play the role of points, and a second invocations in which the roles are reversed. Note that each crossing (between some R_i and L_j) contributes at least one edge to one of the two graphs, and at most four such edges. On the other hand, edges may arise only from intervals that overlap. Again, assuming that Step 4 rejected with very small probability, we infer that the number of pairwise crossing between diagonal segments is small, and it follows that both graphs have relatively few edges. And, again, using Claim 6.6, it follows that

few lines can be omitted from the environment such that all these crossing are eliminated.

In summary, we obtain an environment $\texttt{ENV}^\dagger$ that is ϵ_4-close to $\texttt{ENV}$ such that $\texttt{ENV}^\dagger$ retains all features of $\texttt{ENV}'''$ and in addition contains no crossings between lines. That is, $\texttt{ENV}^\dagger$ consists of *non-crossing* lines that go from the first row to the last row such that each line consists of an initial (possibly empty) diagonal segment followed by a (possibly empty) vertical segment.

Inferring from the success of Step 5. The only aspect in which $\texttt{ENV}^\dagger$ may be inconsistent with the ("moving object") evolution rule is that objects may stop with no good reason (i.e., when their desired direction of movement is not blocked). In terms of the foregoing lines this means that there exists a line with a diagonal segment starting at some position s and ending at position $s+r$ (resp., $s-r$) and another line starting at position $s' < s$ (resp., $s' > s$) and ending at position $s'+r' \in [s, s+r)$ (resp., $s'-r' \in (s-r, s]$) such that the interval $[s'+r', s+r]$ (resp., $[s'-r', s-r]$) is not filled with vertical segments at time t (i.e., $\texttt{ENV}^\dagger_t(p) = \perp$ for some p in the interval). In other words, vertical segments are missing in some locations (i.e., locations that are between the stopping positions of diagonal segments that have overlapping horizontal projections). We shall first show that the success of Step 5 implies that the number of such missing segments in $\texttt{ENV}^\dagger$ is relatively small, and next we shall show how to modify $\texttt{ENV}^\dagger$ so as to eliminate them, while maintaining all other features of $\texttt{ENV}^\dagger$ and thus obtaining an environment that is consistent with the ("moving object") evolution rule.

We shall deal separately with objects moving to the right and with objects moving to the left, capitalizing on the fact that these movements do not cross and that our modifications are internal to the intervals of moving objects. We shall focus on lines that correspond to objects moving to the right, and lines that correspond to objects moving to the left can be dealt in exactly the same way.

We start with a rough overview of the argument. For a position s such that $\texttt{ENV}^\dagger(s) = 1$, we denote by $r(s)$ the stopping time (in $\texttt{ENV}^\dagger$) of the object whose initial position is s. Thus, the movement interval of this object is $[s, s+r(s))$. We cluster the moving objects into buckets according to the values of the start and end points of their movement intervals, and dispose of all small buckets (i.e., omit all the lines that correspond to objects placed in small buckets). Furthermore, assume for a moment that all buckets correspond to long enough intervals (i.e., contain objects with a sufficiently long movement interval). We consider a graph in which the remaining buckets are vertices, and edges connect buckets that correspond to intervals that overlap. Relying on the success of Step 5, we infer that there are few missing verticals in the intervals that correspond to these connected components (or buckets).

Recall that this argument ignores short buckets (i.e., buckets that contain lines with a short movement interval). It is tempting to modify the corresponding lines into vertical lines, but this may create crossings with a large number of long diagonal segments (which may be close by). So a slightly more careful treatment is required here; specifically, we extend the treatment of long intervals to short intervals that are at the proximity of long intervals, and turn short diagonals into verticals when they are not at the proximity of a long interval.

Another level of complication arises from the fact that Step 5 refers only to the approximate stopping time of sampled objects. For such an object, starting at a position s, we do not have its actual stopping time $r(s)$, but rather we only know that it resides in a relatively small interval; that is, $r(s) \in (r^-(s), r^+(s)]$, where we may assume that $r^+(s) - r^-(s)$ is small. (Recall that $r^-(s), r^+(s) \in R$ are such that the object s still moves at time $r^-(s)$ but stands at time $r^+(s)$.) In

what follows, we shall use the (diagonal segments) of sampled lines to create a "skeleton" according to which we decide which lines to remove and which to modify so as to get a legal environment (according to the evolution rule) that is close to $\texttt{ENV}^\dagger$.

Consider clustering all lines into buckets according to the starting and ending points of their diagonal segments (completely vertical lines are not clustered, and will not be modified). For every $j, k \in [1/\epsilon_5]$, we place in bucket $B_{j,k}$ all lines with a diagonal that starts in the interval $((j-1)\epsilon_5 n, j\epsilon_5 n]$ and ends in the interval $((k-1)\epsilon_5 n, k\epsilon_5 n]$. Note that $j \leq k$ must hold, and equality is possible.

We shall say that a bucket is small if the number of lines belonging to the buckets is smaller than $\epsilon_5^3 n$. Otherwise it is large. (As stated above, we shall omit all small buckets, and omit all lines that reside in them.)

For the sake of the analysis, we partition the sample S into two equal-size parts, S^1 and S^2. (This is actually a partition of the sample S_5 that is "designated" for this step, so that it is independent of the samples that were selected in previous steps.) From this point on, we shall make the following assumptions, which hold with high probability (over the choice of the sample S_5).

1. For each large bucket $B_{j,k}$, consider sorting the lines in $B_{j,k}$ according to their starting position and partitioning the lines into consecutive subsets of size $\epsilon_5^3 n/4$ (more precisely, of size at least $\epsilon_5^3 n/4$ and at most $\epsilon_5^3 n/2$). The assumption is that the sample S^1 includes at least one (starting position of a) line from each such subset.

2. For each $s \in S^1$ that corresponds to the starting position of a moving object in $\texttt{ENV}^\dagger$, we have that $r^+(s) - r^-(s) \leq \epsilon_5^4 t$. (where the notations $r^+(s)$ and $r^-(s)$ were introduced in Step 5).

   Actually, this assumption refers to the sample R, and holds with very high probability.

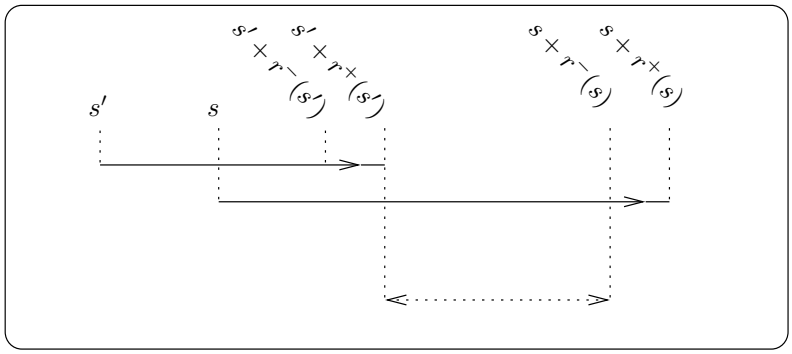


Figure 17: *The intervals considered in Assumption 3. The two arrows depict the actual movement of two objects, starting at locations $s' < s$. The actual stopping times are within the intervals $(s'+r^-(s'), s'+r^+(s'))$ and $(s+r^-(s), s+r^+(s))$, respectively. The interval $[s'+r^+(s'), s+r^-(s)+1]$ is depicted by a dotted double-arrow.*

3. Consider the union U of all intervals $[s'+r^+(s'), s+r^-(s)+1]$ for $s, s' \in S^1$ such that $s' < s$ and $s'+r^-(s')+1 \geq s$. (See Figure 17.) If the number of points $p \in U$ such that $\texttt{ENV}_t^\dagger(p) = \perp$ is greater than $\epsilon_5 n$, then S^2 contains at least one such point.

In addition, as in previous steps, we assume that for all points queried by the algorithm, ENV agrees with ENV†. We next show that conditioned on the above assumptions as well as on the assumption that Step 5 passes successfully, we can transform ENV† into a legal environment by making $O(\epsilon_5 n^2)$ modifications. These modifications are performed on right-moving objects only (and the modifications performed on left-moving objects are analogous), whereas objects that stand from time 0 are not modified. (We may deal separately with right-moving objects and left-moving objects due to the non-crossing property of ENV†.)

We perform the modifications in two stages. In the first stage we remove few lines and transform some lines with short diagonal segments into vertical lines. We show that at the end of this stage the number of positions that correspond to empty gaps at time t between standing objects is small. In the second stage we show how to add lines so as to fill these gaps. In what follows we identify lines with their starting position (i.e., the initial position of the corresponding object). In particular, when we refer to the bucket that a point s belongs to, we mean the bucket that the line whose starting position is s belongs to.

Stage I. We start by removing all lines belonging to small buckets. By the definition of small buckets and the fact that there are less than $1/\epsilon_5^2$ buckets, the total number of lines removed is at most $(1/\epsilon_5)^2 \cdot \epsilon_5^3 n = \epsilon_5 n$, and the number of modification due to these removals is at most $\epsilon_5 nt$.

From this point on we will assume that all (non-empty) buckets are large and refer only to these buckets. We say that a bucket is long if $k \geq j+2$, otherwise (i.e., $k \in \{j, j+1\}$) it is short. Note that intervals that belong to the same long bucket must overlap on at least $\epsilon_5 n$ points: Indeed, if $s \in B_{j,k}$ (and $k \geq j+2$), then $s \in [(j-1)\epsilon_5 n, j\epsilon_5 n]$ and $s + r(s) \in [(k-1)\epsilon_5 n, k\epsilon_5 n]$, which implies $[s, s+r(s)] \subseteq [j\epsilon_5 n, (k-1)\epsilon_5 n]$.

Next, we define a graph $H = (P, E)$ such that the vertex set P contains some points in S^1 from each (non-empty) buckets and edges represent overlapping intervals between these points. Specifically, for each long bucket $B_{j,k}$, the set P contains two points in $B_{j,k} \cap S^1$: the smallest $s \in B_{j,k} \cap S^1$ and the largest such s. By Assumption 1, (more than) two such (different) points exist for each bucket. For each short bucket, P contains exactly one point in S^1 from each $\Theta(\epsilon_5^3 n)$-sized subset as defined in Assumption 1. It follows that $|P| = O(1/\epsilon_5^3)$. We put an edge between s and $s' < s$ in P if (and only if) $s' + r^-(s') + 1 \geq s$. (By the definition of $r^-(\cdot)$ this means that $s' + r(s') \geq s$, implying that at time t there should be standing objects in all points $p \in [s' + r(s'), s + r(s)]$.) Observe that for each long bucket there is an edge between every pair of (sample) points in the bucket (and so the two selected sample points from the same long bucket belong to the same connected component in H).

Consider the connected components in H. We say that a line (starting at) s that belongs to a long bucket $B_{j,k}$ (but is not necessarily in the sample S^1) is assigned to a connected component C in H if the points from $B_{j,k} \cap S^1$ belong to C. Recall that such a long bucket has two points in H, and these two points are necessarily in the same connected component. In contrast, short buckets may have $O(1/\epsilon_5)$ points in H, and these points need not belong to the same connected component. Hence, for each short bucket $B_{j,k}$, we say that a line (starting at) s that belongs to $B_{j,k}$ is assigned to a connected component C in H if the sampled point $s' \in S^1 \cap B_{j,k}$ closest to s belongs to C. Denote the lines assigned to C by $A(C)$.

For each connected component C in H, let $s_{\min}(C)$ be the point $s \in C$ for which s is minimized, and $s_{\max}(C)$ be the point $s \in C$ for which s is maximized. Note that $s_{\min}(C)$ and $s_{\max}(C)$ do not necessarily belong to the same bucket. Loosely speaking, we now remove lines that end after the endpoint point of the line $s_{\max}(C)$ while starting before $s_{\min}(C')$ for all components that start after

C. Specifically, we remove lines as follows.

- Let $C_1, \ldots, C_m$ be an ordering of the connected components according to $s_{\min}(\cdot)$. By the definition of H, we have that $s_{\max}(C_i) + r^-(s_{\max}(C_i)) + 1 < s_{\min}(C_{i+1})$ (since there is no edge between $s_{\max}(C_i)$ and $s_{\min}(C_{i+1})$).

  We shall say that a connected component C_i is short if all lines that are assigned to it (i.e., all lines in $A(C)$) are short. Otherwise C_i is long.

- The default is that for each $i \in [m-1]$, we remove all lines s' in $C_i \cup C_{i+1}$ such that $s' + r(s') > s_{\max}(C_i) + r^-(s_{\max}(C_i)) + 1$ and $s' < s_{\min}(C_{i+1})$. (Indeed we may remove $s_{\max}(C_i)$ itself). The exception is for the case that both C_i and C_{i+1} are short. In this case we do not remove the lines "between" these two connected components.

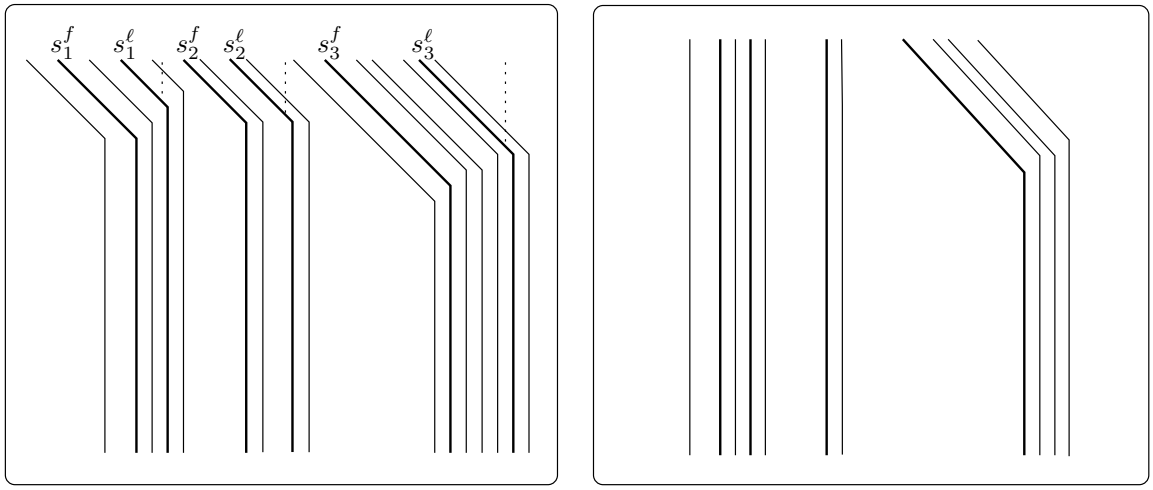


Figure 18: *An illustration for modifications performed in Stage I of the analysis of Step 5. The left image shows (part of) the environment* $\texttt{ENV}^\dagger$ *before lines are removed or "straightened", whereas the right image shows the environment after these changes. The illustration shows three connected components* C_j*, where* C_1 *and* C_2 *are short, while* C_3 *is long. The notation* s_j^f *stands for* $s_{\min}(C_j)$*, and* s_j^ℓ *stands for* $s_{\max}(C_j)$*. Lines starting at the corresponding six positions are bold. The dotted lines indicate the positions* $s_j^\ell + r^-(s_j^\ell) + 1$*. Since* C_1 *and* C_2 *are both short, no lines are removed "between" them, while since* C_3 *is long all lines "between"* C_2 *and* C_3 *are removed, as well as the lines "at the end" of* C_3*. Finally, all (remaining) lines in the short buckets* C_1 *and* C_2 *are turned into vertical lines.*

  In addition, if C_m is a long bucket, then we remove all points s' (in C_m) such that $s' + r(s') > s_{\max}(C_m) + r^-(s_{\max}(C_m)) + 1$, and if C_1 is a long bucket, then remove all points s' (in C_1) such that $s' < s_{\min}(C_1)$.

Using Assumption 1, we show that the total number of lines removed is $O(\epsilon_5 n)$, and it follows that the contribution to the number of modifications is $O(\epsilon_5 nt)$. This is shown by charging each omitted line $s' \in C_i \cup C_{i+1}$ either to the bucket containing $s_{\max}(C_i)$ or to the bucket containing $s_{\min}(C_{i+1})$. If one of these buckets is long, then s' must belong to the same $\Theta(\epsilon_5^3 n)$-sized extreme subset of the bucket (i.e., to the last subset of $B_{j,k}$ in case $s_{\max}(C_i)$ belongs to $B_{j,k}$ that is long, and to the first

subset of $B_{j,k}$ in case $s_{\min}(C_{i+1})$ belongs to $B_{j,k}$ that is long). If both buckets are short, then these buckets may be used as basis for removing lines for at most two values of $i \in [m]$, and for each value of i the same $\Theta(\epsilon_5^3 n)$-sized subset of the bucket is used (although it need not be an extreme one). Hence, each value of i causes $O(\epsilon_5^3 n)$ omissions, whereas $m < 1/\epsilon_5^2$.

We now transform all lines that are assigned to short connected components into vertical lines. Given our rules for removing lines, no crossings are created now (between lines in short connected components and lines in long connected components). Each modified line incurs a cost of $3\epsilon_5 n$, since it belongs to a short bucket, and so the total number of modifications due to this transformation is $O(\epsilon_5 n^2)$. For an illustration of the removal of lines and the changes in lines belonging to short buckets, see Figure 18.

Let the resulting environment be denoted by $\texttt{ENV}^{\ddagger}$, and let M denote the set of *missing* positions p. That is, for each $p \in M$ there exist $s' < s$ for which $\texttt{ENV}^{\ddagger}(s') = \texttt{ENV}^{\dagger}(s) = +1$ while $s' + r(s') < p < s + r(s)$ and $\texttt{ENV}^{\ddagger}_t(p) = \perp$. Observe that based on the definition of H and the lines we removed and modified, for each $p \in M$ there exists a connected component C such that $s_{\min}(C) + r^-(s_{\min}(C)) + 1 < p < s_{\max}(C) + r^+(s_{\max}(C))$.

We claim that $|M| = O(\epsilon_5 n)$. To verify this, for each long connected component C (whose lines were not turned into vertical lines), consider a shortest path from $s_{\min}(C)$ to $s_{\max}(C)$, denoted $s^1(C), \ldots, s^\ell(C)$, where $\ell \leq |P| < 1/\epsilon_5^3$. We shall say that a point p is covered by this path if for some $j \in [\ell - 1]$ it holds that $p \in [s^j(C) + r^+(s^j(C), s^{j+1}(C) + r^-(s^j(C)) + 1]$. By the definition of H, Assumption 3, and the premise that Step 5 completed successfully, the number of points $p \in M$ that belong to any interval $[s^j(C) + r^+(s^j(C)), s^{j+1}(C) + r^-(s^{j+1}(C)) + 1]$ (i.e., for any C and any j) is at most $\epsilon_5 n$. On the other hand, by Assumption 2, for each C and each j, the number of points $p \in [s^j(C) + r^-(s^j(C)), s^j(C) + r^+(s^j(C))]$ is at most $2\epsilon_5^4 n$. Hence, $|M| \leq \epsilon_5 n + |P| \cdot 2\epsilon_5^4 = O(\epsilon_5 n)$.

Stage II. In this stage we add lines so as to get a legal environment. First, for each long connected component C, let $s'_{\max}(C)$ denote the maximum s' (not necessarily in S) that are assigned to C in $\texttt{ENV}^{\ddagger}$. (Recall that we may have removed $s_{\max}(C)$.) We first add a vertical line in position $s_{\max}(C) + r(s_{\max}(C)) + 1$, unless such a line already exists (either vertical or left moving). The total number of lines added is at most $1/\epsilon_5$.

Next, given that the set of positions M is relatively small, we can afford to insert lines in order to have standing objects (i.e., vertical segments) in these positions, but the question is whether we can do so in a legal manner. The simple case corresponds to a gap (among the vertical segments) such that the corresponding diagonal segments are far enough from one another (see the gap indicated by (i) in Figure 19). In this case we just insert a line (consisting of a vertical segment and a diagonal segment) in this gap. The more complicated case (depicted in Figure 19 by the gap marked (ii)) is of a gap (among the vertical segments) such that the corresponding diagonal segments are too close to allow for the insertion of a diagonal segment. In this case we insert a vertical segment at the gap, and continue it diagonally by "taking over" the neighboring diagonal segment (which must be at a very short distance). But then, we should re-route the corresponding vertical segment (i.e., prepend it with a diagonal segment), which we do just as we did when we inserting a new vertical segment (and again there are the same two cases). This means that, in the second case, we enter a sequence of "take overs" such that the changes required for each step (i.e., a single "take over") are very local (and are constant in number), whereas a cost of $O(t)$ (for inserting a vertical segment) is payed at the last step and the number of steps is $O(n)$. Thus, in each case, the closing of a single gap costs $O(t + n)$ changes in the environment. The closing of multiple gaps is done in the direction of movement (i.e., from left to right in Figure 19), while the sequence of "take overs"

that refers to a single gap moves in the opposite direction (i.e., from right to left).

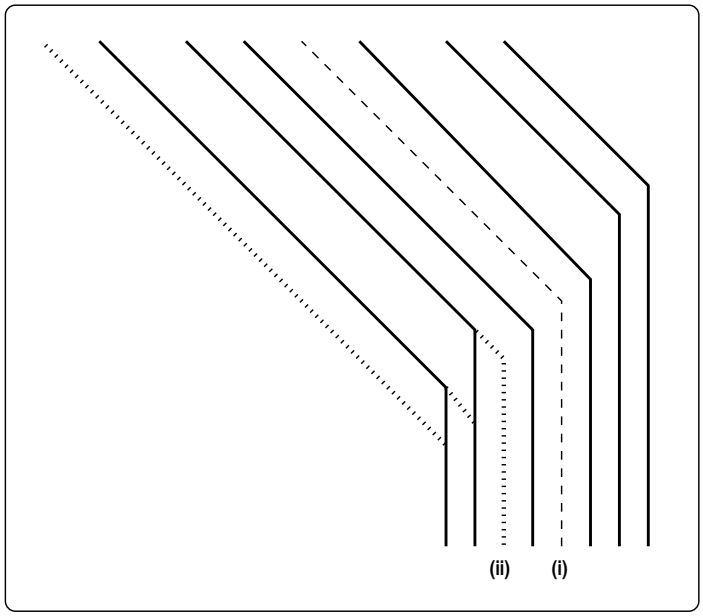


Figure 19: *Filling up the vertical gaps in the analysis of Step 5. The original lines are shown in solid, the dashed line at location (i) is inserted without conflicts, whereas the dotted lines show an insertion at location (ii) and its propagated corrections.*

To summarize, the said gaps (i.e., the position in M) can be eliminated at a cost of $O(|M| \cdot n)$ modifications. Since $t = \Omega(\epsilon n)$ and $|M| = O(\epsilon_5 \cdot n)$, we obtain an environment $\texttt{ENV}^\star$ that is $O(\epsilon_5/\epsilon))$-close to ENV and is consistent with the ("moving object") evolution rule, where the argument relies on picking ϵ_5 such that $\epsilon_4 = \text{poly}(\epsilon_5)$.

Conclusion: Deriving Theorem 1.10 for the case of $t \geq \epsilon n/2$. Picking $\epsilon_5 = \epsilon^2/O(1)$ and $\epsilon_i = \text{poly}(\epsilon_{i+1})$ for every $i = 4, ..., 1$, the entire argument goes through for $t \in [0.5\epsilon n, n]$. Specifically, if the test (described at the beginning of this section) accepts ENV with probability at least 1/3, then ENV is ϵ-close to being consistent with the ("moving object") evolution rule. The reason for the constraint that $t \in [0.5\epsilon n, n]$ is due to the fact that we used this condition in our analysis (in Step 3). Our aim is to establish this result also for the other cases.

Dealing with the case of $t > n$ is quite easy; actually, the current tester will do. The key observation is that, after n evolution steps, each of the moving objects either exited the environment or got stuck in a standing state (within the environment). Hence, we need only apply the current tester for the first n steps of the evolution, and may apply a rather simple tester for the remaining $t - n$, where the latter tester merely checks that objects that stand at time n continue standing at any time in $[n, t]$. Actually, the current test essentially performs the latter checking (where for $t \gg n/\epsilon$ it actually checks that objects that stood at time approximately ϵt continue to stand at later times). Also note that when $t > 2n/\epsilon$, it suffices to perform only the latter test (since the first n evolution steps can be modified, in a trivial manner, to fit the standing pattern of the last $t - n$ steps).

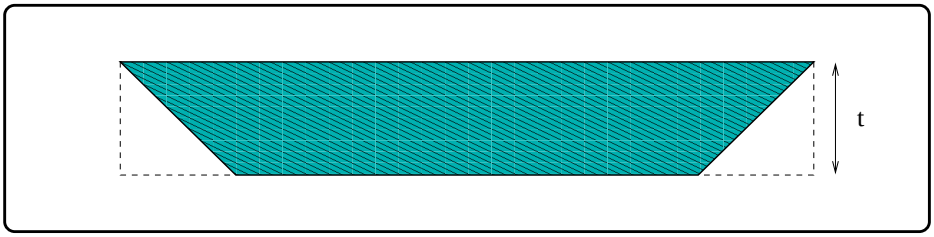


Figure 20: *The trapezoid environments used for the case of $t < \epsilon n/2$.*

Conclusion: Deriving Theorem 1.10 for the case of $t < \epsilon n/2$. We handle this case by first observing that our treatment of the case of a t-by-$(2t/\epsilon)$ rectangular environments (i.e., $n = 2t/\epsilon$) extends to trapezoid environments that are obtained by chopping off two right-angle triangles as shown in Figure 20. When applying the tester to such trapezoid environments, we ignore all queries made to the triangles that were chopped off, just as we ignored locations that are outside the rectangular environments that were considered so far. Now, when testing a (rectangular) t-by-n environment, where $t < \epsilon n/2$, we cut the rectangle into consecutive t-by-$(2t/\epsilon)$ environments (see Figure 21) and apply the "trapezoid tester" to a sample of $O(1/\epsilon)$ of these trapezoid environments.

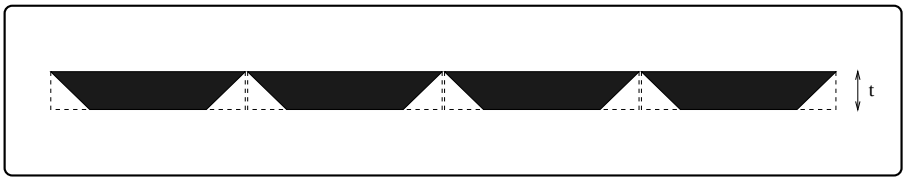


Figure 21: *Tiling a rectangle with trapezoids (for the case of $t < \epsilon n/2$).*

Note that any t-by-n environments that evolves according to the moving-objects rule is accepted by this tester, with probability at least $2/3$, because the evolution within each trapezoid is according to the rule. The latter fact follows because the movement within each trapezoid involve only objects that started within this trapezoid. (This is the reason that we chopped off the triangles.) On the other hand, if the t-by-n environment is ϵ-far from any environment that evolves according the moving-objects rule, then at least $\epsilon/4$ fraction of the trapezoids are $\epsilon/4$-far from evolving according to the said rule, since only an $\epsilon/2$ fraction of the area is lost by the chopped triangles. This completes the proof of Theorem 1.10 also for this case; that is, we have proved the following.

Theorem 6.7 (Theorem 1.10, restated): *There exists a time-conforming oracle machine of* (total) *time complexity* poly$(1/\epsilon)$ *that tests the consistency of evolving environments with the fixed-speed movement of objects in one dimension, where colliding objects stop forever. Furthermore, the tester is nonadaptive, but it has two-sided error.*

6.1.2 On the complexity of one-sided error testers

The foregoing tester (as asserted in Theorem 6.7) has two-sided error. As stated at the beginning of Section 6.1.1, this is unavoidable for testers of query complexity that meets the claim of Theorem 6.7 (i.e., having complexity that does not depend on n).

Theorem 6.8 (lower bound on one-sided testers): *Any nonadaptive* one-sided error *tester for the consistency of environments of the form* $\texttt{ENV} : \{0,1,...,n\} \times [n] \to \{-1,0,1,\perp\}$ *with the moving object evolution rule has* (total) *query complexity* $\Omega(\sqrt{n})$.

Before proving Theorem 6.8, we discuss its implications. First note that Theorem 6.8 implies that *any time-conforming one-sided error tester for the foregoing property has* (total) *query complexity* $\Omega(\log n)$. The latter lower bound holds even for adaptive (one-sided error) testers that are not time-conforming. We also note that the tester we used for proving Theorem 6.7 is actually nonadaptive.

We do not know whether the logarithmic query complexity lower bound is tight for general *time-conforming one-sided error testers*, but it is certainly tight for arbitrary testers that are not time-conforming. That is, an oracle machine that is *not time-conforming* can test such environments with one-sided error probability and total time complexity $\text{poly}(\epsilon^{-1} \log n)$. For example, one may replace Step 3 in the foregoing tester by sampling $\text{poly}(\epsilon^{-1})$ start positions and verifying that if a diagonal segment starts in some position at time 0 then it either exits $[1,n]$ or turns into a vertical segment (which ends at time n). Similarly, one should check that vertical segments that end at time n either start as vertical lines (at time 0) or are the continuation of some diagonal segment. Both checks can be performed by conducting a binary search for the ending/starting locations of the relevant segments, but indeed this binary search (in time) in not time-conforming.

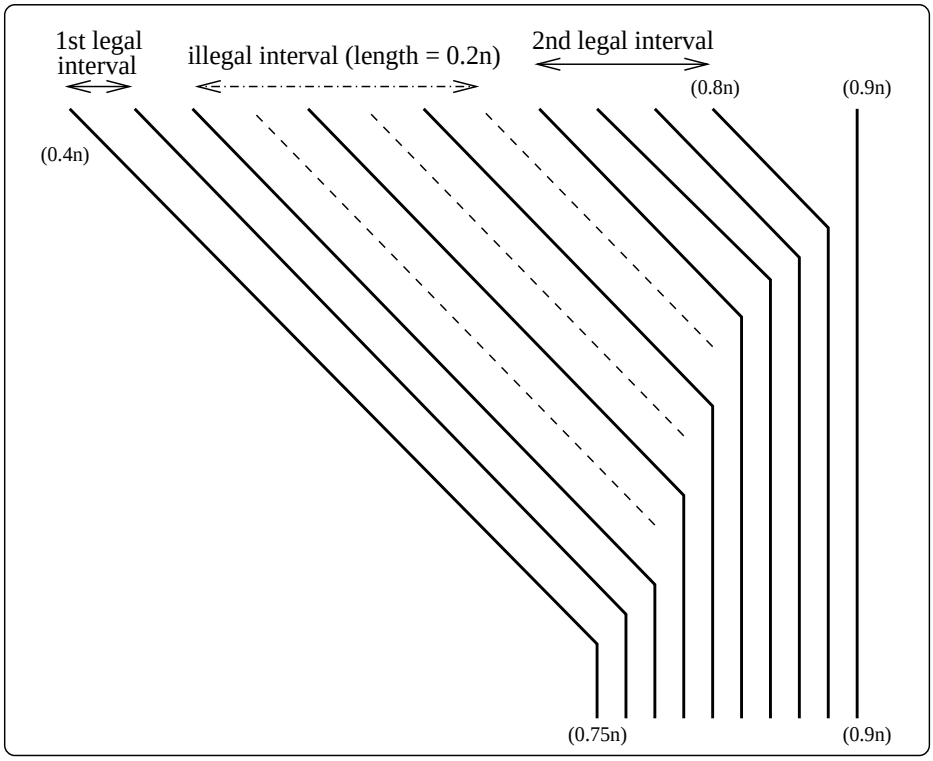


Figure 22: *A generic environment in the family used in the proof of Theorem 6.8. The three dashed lines are illegal (since they have no vertical continuation at the point that they stop). Only a portion of the environment is shown, say* $[0, 0.4n] \times [0.4n, 0.9n]$.

Proof: We prove that any *nonadaptive* one-sided error tester for the moving object evolution rule has (total) query complexity $\Omega(n^{1/2})$. Towards this end, we consider a uniformly chosen environment out of the following family of environments $\mathtt{ENV} : [0,n] \times [n] \to \{-1,0,1,\perp\}$ that are parameterized by an integer $r \in [0.1n]$ (and depicted in Figure 22). Such an environment, denoted $\mathtt{ENV}^{(r)}$, consists of

1. Legal lines starting at any odd location in the interval $[0.4n+1, 0.4n+2r]$. Each of these r lines starts as a diagonal and the i^{th} line turns vertical in location $0.75n+i$.

2. Pairs of lines starting at any odd location in the interval $I_r \stackrel{\text{def}}{=} [0.4n+2r+1, 0.6n+2r]$, called the illegal interval. Hence, we have $0.05n$ such pairs of lines. The first line in each pair is legal, whereas the second has only a diagonal segment. The diagonal segment of the lines of the i^{th} pair stop at column $0.75n+r+i$ (and only the first line turns vertical).

3. Legal lines starting at any odd location in the interval $[0.6n+2r, 0.8n]$. Each of these $0.1n-r$ lines starts as a diagonal and the i^{th} line turns vertical in location $0.8n+r+i$.

4. A vertical line at location $0.9n+1$.

In total, we have $r+0.05n+(0.1n-r) = 0.15n$ legal lines and $0.05n$ illegal lines, where all lines start as diagonals at odd locations in $[0.4n, 0.8n]$. The legal lines end as vertical in locations $[0.75n, 0.9n]$; see Figure 22.

The reader may verify that each of these environments is $\Omega(1)$-far from being consistent with the moving object evolution rule; one way of seeing it is that there are $0.2n$ diagonal segments (each of length at least $0.1n$) but only $0.15n$ vertical segments (each of length at least $0.65n$). Thus, any one-sided error tester that is given access to any of these environments has to find, with high probability, a substructure that is not compatible with the evolution rule. We shall prove that such a substructure cannot be found by a *nonadaptive* oracle machine that makes $o(n^{1/2})$ queries.

We shall show that *for any set of possible queries $Q \subset [n]^2$, with probability at least $1-O(|Q|^2)$ over all possible choices of $r \in [0.1n]$, the answers obtained from $\mathtt{ENV}^{(r)}$ are compatible with some environment that is consistent with the evolution rule.* Hence, for $|Q| = n^{1/2}/O(1)$, with probability at least 0.9, a random execution of any nonadaptive oracle machine on a random $\mathtt{ENV}^{(r)}$ obtains answers that are compatible with a legal environment, and must accept (if it is to constitutes a one-sided error tester). But this implies that there exists an environment $\mathtt{ENV}^{(r)}$ that is accepted by this machine with probability at least 0.9, which contradicts the testing requirement. So it all boils down to proving the foregoing claim (in italics).

The foregoing claim will be proved by showing how to modify $\mathtt{ENV}^{(r)}$ into an environment $\widetilde{\mathtt{ENV}}^{(r)}$ that evolves according to the rule and agrees with $\mathtt{ENV}^{(r)}$ on all queries made (i.e., for every $q \in Q$ it holds that $\widetilde{\mathtt{ENV}}^{(r)}(q) = \mathtt{ENV}^{(r)}(q)$). Actually, the latter claim holds for any Q (of size $\sqrt{n}/O(1)$) and for almost all choices of $r \in [0.1n]$. An obvious case in which this cannot be done is when the set of queries contains the last point (y,z) on an illegal diagonal line as well as its close neighborhood (i.e., $(y,z+1)$, $(y+1,z)$, and $(y+1,z+1)$), which indicates that the diagonal stops with no continuation. Note, however, that the hypothesis that (y,z) is the last point on an illegal diagonal yields a linear equation involving x, y and r; specifically, if $z = 0.75n+r+i$, then $y = 0.35n-r-2i$, which implies that $y+2z = 1.85n+r$. But, for a fixed (x,y), the equality holds with probability (at most) $1/0.1n$, when r is selected at random uniformly in $[0.1n]$.

Unfortunately, the above case is not the only case that may hinder our argument. To see the type of difficulties that arise, consider an attempt to modify $\texttt{ENV}^{(r)}$ into a legal $\widetilde{\texttt{ENV}}^{(r)}$. One observation is that we can remove from $\texttt{ENV}^{(r)}$ any illegal diagonal segment that is not hit by any query. Furthermore, we may can remove from $\texttt{ENV}^{(r)}$ any legal diagonal segment that is not hit by any query, and connect its vertical segment to a neighboring illegal diagonal segment (which was hit by some query). The latter modification is undetectable provided that the corresponding column was not queried (at least not at the point where we modified it). This suggests that we need to avoid a situation in which queries reside both on an illegal diagonal and on the vertical segment that belongs to the legal diagonal that is paired with this illegal diagonal. (Indeed, this is where the expression $|Q|^2$ comes from.)

In general, things are more complicated than that, since there may be queries also on the legal diagonal that is paired with the illegal one. Still, the notions of diagonal and vertical segments (or rather their infinite extensions) that are hit by queries plays a major role; see the sets $D(Q)$ and $C(Q)$ below. In addition, we shall use a process that connects illegal diagonals that were hit by queries to vertical segments, while queuing legal diagonals that were queried and yet their vertical segment was taken by the process. This process will work from right to left (in reverse order to our numbering), and will be captured by the game, which in turn defined a set $G(\cdot)$ of lines that entered the queue. Actually, the set $G(\cdot)$ is a superset of the lines that we should care about, since the queuing in the game is more conservative than the queuing done in the actual process.

We start with some notations. Firstly, let $[[n]] = \{0, 1, ..., n\} = [n] \cup \{0\}$. Next, we define the sets briefly discussed above.

- For any set $Q \subset [[n]] \times [n]$, denote by $D(Q)$ the set of starting positions (i.e., at time 0) of the infinite diagonals on which the elements of Q reside; that is,

$$D(Q) \stackrel{\text{def}}{=} \{s \in [n] : \exists j \in [[n]] \text{ s.t. } (j, s+j) \in Q\}. \tag{12}$$

- For any set $S \subset [n]$, we define a set $G(S)$ by using the following game, which proceeds for $n' \stackrel{\text{def}}{=} 0.1n$ iterations (corresponding to the integers in $[0.4n, 0.8n]$ that are congruent to 1 mod 4). The game is initialized with (a state) $\text{state}_0 = 0$. In the i^{th} iteration, if $S \cap \{0.8n - 4i + 1, 0.8n - 4i + 3\} \neq \emptyset$, then $\text{state}_i \leftarrow \text{state}_{i-1} + 1$ (representing enqueuing an element), else $\text{state}_i \leftarrow \max(0, \text{state}_{i-1} - 1)$ (representing dequeuing an element). Note that $\text{state}_{n'} > 0$ indicates that the queue remains non-empty at the end of the game (which corresponds to a definite failure in our correction process).

  Let $P(S)$ denote the indices of iterations in which the state is positive, and let $G(S)$ contain the elements of $[0.4n, 0.8n]$ that correspond to $P(S)$; that is,

$$G(S) \stackrel{\text{def}}{=} \{0.8n - 4i + 1, 0.8n - 4i + 3 : i \in P(S)\} \tag{13}$$

$$\text{where } P(S) \stackrel{\text{def}}{=} \{i \in [n'] : \text{state}_i > 0\}. \tag{14}$$

Fact 6.8.1 *Let $1 \leq i_1 \leq i_2 \leq 0.1n - |S|$ be integers such that $S \subseteq [0.8n - 4i_2 + 1, 0.8n - 4i_1 + 3]$. Then, $G(S) \subseteq [0.8n - 4(i_2 + |S|) + 1, 0.8n - 4i_1 + 3]$.*

Proof: First note that for every $i < i_1$, it holds that $i \notin P(S)$. Next note that $\text{state}_{i_2} \leq |S|$, since whenever state_i is incremented some (different) element of S is charged for it, whereas for every $i > i_2$ it holds that $\text{state}_i = \max(0, \text{state}_{i-1})$. Hence, $\text{state}_{i_2+|S|} = 0$ and $P(S) \subseteq [i_1, i_2 + |S| - 1]$ follows. ■

Fact 6.8.2 *For any $S \subseteq [n]$, it holds that $|G(S)| \leq 4|S|$.*

Proof: We first note that for every $i \in [n']$ either $\text{state}_i = \text{state}_{i-1}+1$ or $\text{state}_i = \max(0, \text{state}_{i-1}-1)$. Now, if $i \in P(S)$ and $\text{state}_i \neq \text{state}_{i-1} + 1$, then $\text{state}_i = \text{state}_{i-1} - 1$ must hold (since $i \in P(S)$ implies $\text{state}_i > 0$). Next note that $|\{i \in [n'] : \text{state}_i = \text{state}_{i-1} - 1\}|$ equals $|\{i \in [n'] : \text{state}_i = \text{state}_{i-1}+1\}| \leq |S|$, where the inequality follows since $\text{state}_i = \text{state}_{i-1}+1$ implies that $S \cap \{0.8n - 4i + 1, 0.8n - 4i + 3\} \neq \emptyset$. Hence $|P(S)| \leq 2|S|$, and the fact follows. ■

- For any $r \in [0.1n]$ and odd $s \in [0.4n + 2r + 1, 0.6n + 2r]$, define $e_r(s)$ to be the ending (or stopping) column of the diagonal segment that starts at position s; that is, $e_r(s) = 0.75n + r + \lceil (s - (0.4n + 2r + 1))/4 \rceil$, which equals $0.65n + \lfloor r/2 \rfloor + \lceil s/4 \rceil$. Indeed, for any $i \in [0.05n]$, it holds that $e_r(0.4n + 2r + 4i + 1) = e_r(0.4n + 2r + 4i + 3) = 0.65n + r + i$, reflecting the fact that the corresponding pair of diagonals end at the same column.

  For odd $s \in [0.4n+1, 0.4n+2r]$, we define $e_r(s) = 0.75n+\lfloor (s - 0.4n)/2 \rfloor = 0.55+\lfloor s/2 \rfloor$, which is indeed the ending (or stopping) column of the diagonal segment that starts at position s. (Indeed, e_r is not defined for other values.)

  Abusing notation, for a set $S \subseteq [0.8n]$, we define $e_r(S) \stackrel{\text{def}}{=} \{e_r(s) : s \in S \cap [0.4n + 1, 0.8n]\}$, which means that the elements of $S \setminus [0.4n + 1, 0.8n]$ are ignored.

- For any $Q \subset [n]^2$, denote by $C(Q)$ the set of columns that are contain a point in Q; that is, $C(Q) \stackrel{\text{def}}{=} \{c \in [n] : \exists (j, c) \in Q\}$.

The following claim upper bounds the probability that the set of the stopping columns of queried diagonals that start at the illegal interval (i.e., $I_r = [0.4n+2r+1, 0.6n+2r]$) is "related" to the set of columns that contain some query, where the relation is the one that arises from $G(\cdot)$ and $e_r(\cdot)$. That is, we consider the intersection of the sets $e_r(G(D(Q) \cap I_r))$ and $C(Q)$. Note that the first set is a random variable, which depends on the random $r \in [0.1n]$, whereas the second set is fixed.

Actually, we will augment the above condition (i.e., $e_r(G(D(Q) \cap I_r)) \cap C(Q) = \emptyset$) in two ways: Firstly, we shall augment the set of columns with the column $0.75n$, which is equivalent to requiring that the set of diagonals to which e_r is applied does not contain $0.4n + 1$. (This reflects the queue being empty at the end of the game.) Secondly, we shall augment the set $G(D(Q) \cap I_r)$ with the $|Q|$ last diagonals of the first legal region (i.e., the odd integers in $[0.4n+2r-2(|Q|-1), 0.4n+2r]$). The reason for these augmentation will be clarified in the proof of Claim 6.8.4.

Claim 6.8.3 *Recall that $I_r = [0.4n + 2r + 1, 0.6n + 2r]$ and let $A_{r,s} = [0.4n + 2r - 2(s - 1), 0.4n + 2r] \cap \{2i - 1 : i \in \mathbb{N}\}$. For any set $Q \subset [n]^2$, it holds that*

$$\Pr_{r \in [0.1n]} \left[e_r(G(D(Q) \cap I_r) \cup A_{r,|Q|}) \cap (C(Q) \cup \{0.75n\}) \neq \emptyset \right] < \frac{90(|Q| + 1)^2}{n} .$$

Proof: By Fact 6.8.1, it holds that $G(D(Q) \cap I_r) \cup A_{r,|Q|}$ equals $(G(D(Q) \cap I_r) \cap I_r) \cup A_{r,|Q|}$, which is contained in $(G(D(Q)) \cap I_r) \cup A_{r,|Q|}$. Hence:

$$\begin{aligned}
&\Pr_{r\in[0.1n]}\left[e_r(G(D(Q)\cap I_r)\cup A_{r,|Q|})\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \\
&\quad\leq \Pr_{r\in[0.1n]}\left[e_r(G(D(Q))\cap I_r)\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \qquad (15)\\
&\qquad+\Pr_{r\in[0.1n]}\left[e_r(A_{r,|Q|})\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \qquad (16)
\end{aligned}$$

We start with Eq. (16). By the definition of $e_r(\cdot)$, it holds that $e_r(A_{r,|Q|}) = \{0.55n + r - |Q|, ..., 0.55n + r\}$. Hence Eq. (16) reduces to

$$\begin{aligned}
&\Pr_{r\in[0.1n]}\left[\{0.55n+r-|Q|,...,0.55n+r\}\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \\
&\quad\leq \sum_{i\in[[|Q|]]}\ \sum_{j\in(C(Q)\cup\{0.75n\})}\Pr_{r\in[0.1n]}\left[0.55+r-i=j\right] \\
&\quad\leq (|Q|+1)\cdot(|C(Q)|+1)\cdot\frac{1}{0.1n}
\end{aligned}$$

which is upper bounded by $10 \cdot (|Q| + 1)^2/n$. Turning to Eq. (15), we note that for every $s \in D(Q)$ such that $G(s) \in I_r$ it holds that $e_r(s) = 0.65n + \lfloor r/2 \rfloor + \lfloor s/4 \rfloor$. Hence, Eq. (15) is upper bounded as follows:

$$\begin{aligned}
&\Pr_{r\in[0.1n]}\left[e_r(G(D(Q))\cap I_r)\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \\
&\quad\leq \sum_{i\in G(D(Q))}\ \sum_{j\in(C(Q)\cup\{0.75n\})}\Pr_{r\in[0.1n]}\left[i\in I_r\ \&\ e_r(i)=j\right] \\
&\quad\leq \sum_{i\in G(D(Q))}\ \sum_{j\in(C(Q)\cup\{0.75n\})}\Pr_{r\in[0.1n]}\left[0.65n+\lfloor r/2\rfloor+\lfloor i/4\rfloor=j\right] \\
&\quad\leq |G(D(Q))|\cdot(|C(Q)|+1)\cdot\frac{2}{0.1n} \\
&\quad\leq \frac{20}{n}\cdot 4|Q|\cdot(|Q|+1),
\end{aligned}$$

where the last inequality uses Fact 6.8.2. The claim follows. ■

Next, we show that if the event referred to in Claim 6.8.3 does not occur (i.e., if $e_r(G(D(Q) \cap I_r))$ does not intersect $C(Q) \cup \{0.75n\}$), then the answers on the queries Q obtained from $\mathtt{ENV}^{(r)}$ are consistent with the evolution rule (or rather are compatible with some environment that evolves according to this rule). For a fixed choice of r, we shall also use the notation $\mathtt{ENV}^{(r)}_Q$ to denote the restriction of the environment $\mathtt{ENV}^{(r)}$ to Q. In other words, $\mathtt{ENV}^{(r)}_Q$ is determined by the answers to the queries in Q when the environment is $\mathtt{ENV}^{(r)}$.

Claim 6.8.4 *Let $A_{r,s}$ be as in Claim 6.8.3. If $e_r(G(D(Q) \cap I_r) \cup A_{r,|Q|}) \cap (C(Q) \cup \{0.75n\}) = \emptyset$, then $\mathtt{ENV}^{(r)}_Q$ is consistent with a legal environment of the moving object evolution rule.*

Proof: Given a partial environment $\mathtt{ENV}^{(r)}_Q$ that satisfies $e_r(G(D(Q)\cap I_r)\cup A_{r,|Q|})\cap(C(Q)\cup\{0.75n\}) = \emptyset$, we show how to extend it to a legal environment $\widetilde{\mathtt{ENV}}^{(r)}$ such that $\widetilde{\mathtt{ENV}}^{(r)}_Q = \mathtt{ENV}^{(r)}_Q$. To this end

we construct a matching between diagonals and vertical lines. If a diagonal starting in position $(0, s)$ is matched to a vertical line that ends in position (n, e), where necessarily $e > s$, then this means that in $\widetilde{\texttt{ENV}}^{(r)}$ there is a diagonal that starts in position $(0, s)$ and turns into a vertical line in position $(e - s, e)$.

In $\widetilde{\texttt{ENV}}^{(r)}$, as in $\texttt{ENV}^{(r)}$, there is a vertical line extending from $(0, 0.9n)$ to $(n, 0.9n)$. In general, in $\widetilde{\texttt{ENV}}^{(r)}$, as in $\texttt{ENV}^{(r)}$, there will be vertical lines ending in all positions (n, e) for every $e \in [0.75n, 0.9n]$, where the difference between $\texttt{ENV}^{(r)}$ and $\widetilde{\texttt{ENV}}^{(r)}$ may be in the starting positions of these vertical lines. Each of these vertical lines will be matched to a diagonal line starting in the first row, where the matching is "non-crossing". That is, if the vertical line in column e is matched to the diagonal starting in position $(0, s)$, then for every $e' < e$, the vertical line in column e' is matched to a diagonal starting in position $(0, s')$ where $s' < s$. The starting positions of the vertical lines will be determined by the meeting points with the diagonals they are matched to. If, for an odd position $s \in [0.4n, 0.8n]$ we get that $(0, s)$ is not matched to any vertical line, then in $\widetilde{\texttt{ENV}}^{(r)}$ there will be no diagonal starting in position $(0, s)$. Thus, we ensure that $\widetilde{\texttt{ENV}}^{(r)}$ is a legal environment.

The matching is constructed iteratively as follows, going "backwards" from the vertical line in column $0.9n - 1$. In iteration i we match the vertical line in column $e_i = 0.9n - i$ to the diagonal starting in position $(0, s_i)$, where s_i is determined as follows. For $i = 1$, we let s_1 be the largest odd value $s \leq 0.8n$ (so that there is a legal diagonal starting in position $(0, s_1)$ in $\texttt{ENV}^{(r)}$, and this diagonal turns into a vertical line in column e_1). To determine s_i for $i > 1$ we maintain a queue, where the queue is initially empty. In general, the queue will only contain (temporarily) odd indices s for which the following holds:

1. The diagonal starting in position $(0, s)$ in $\texttt{ENV}^{(r)}$ ends in a column e (in $\texttt{ENV}^{(r)}$) that was matched to some diagonal starting in position $(0, s')$ for some $s' > s$; and

2. there is a query in Q that resides on the diagonal starting at $(0, s)$.

Note that s may correspond to either a legal or an illegal diagonal in $\texttt{ENV}^{(r)}$.

As long as $s_{i-1} > 0.6n+2r+2$ (which means that we are in the second legal interval), the vertical line in column e_i is matched to the (legal) diagonal starting in position $(0, s_i)$, where $s_i = s_{i-1} - 2$, and so that $\widetilde{\texttt{ENV}}^{(r)}$ is the same as $\texttt{ENV}^{(r)}$ in this region. In this case, the queue remains empty. Once $s_{i-1} \leq 0.6n + 2r + 2$, which means that we enter the illegal region, and until we exit it, we proceed as follows, where in each iteration i we assign a value to s_i (and match e_i to s_i).[36]

- If the queue is empty at the start of iteration i, then there are several cases (which are depicted in Figure 23).

    1. The following two cases corresponds to lack of a new query on the relevant diagonal (determined by the case).
        (a) If s_{i-1} corresponds to a legal diagonal and there is no query on the illegal diagonal that corresponds to $s_{i-1} - 2$, then we set $s_i = s_{i-1} - 4$. (See Case 1a in Figure 23.)
        (b) Similarly, if s_{i-1} corresponds to an illegal diagonal and there is no query on the illegal diagonal that corresponds to $s_{i-1} - 4$, then we set $s_i = s_{i-1} - 6$.

[36] Recall that in the illegal region the diagonals come in pairs. Each pair consists of one legal diagonal and one illegal diagonals, which end in the same column, where the legal diagonal starts two positions before the illegal one.

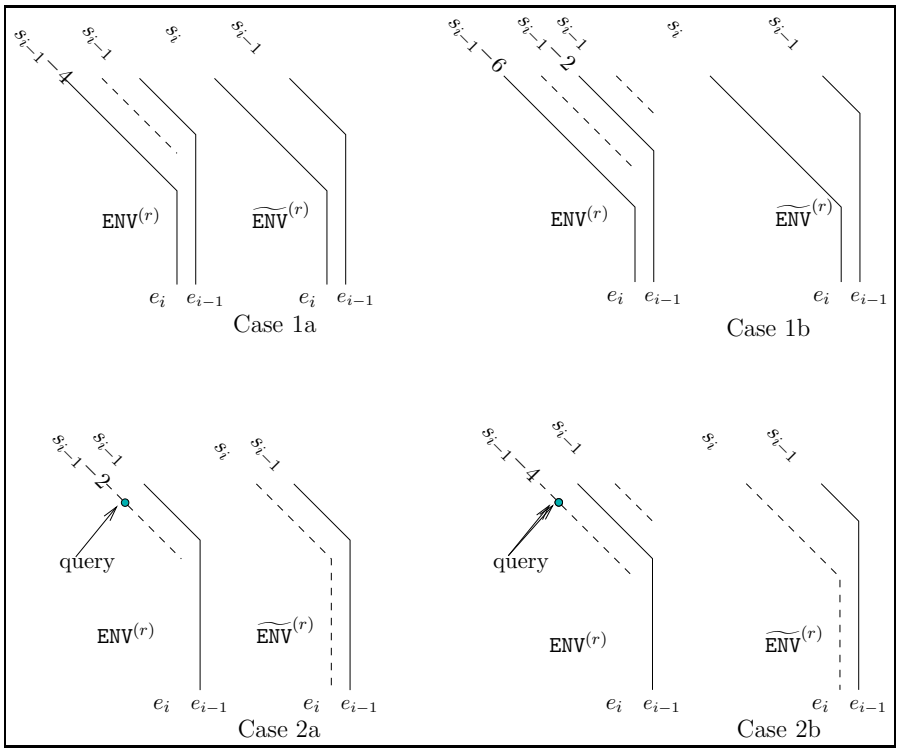


Figure 23: *Detail for the proof of Claim 6.8.4. An illustration for the way the matching is constructed in the illegal region when the queue is empty. As in Figure 22, in the images for* $\mathtt{ENV}^{(r)}$*, the dashed lines represent illegal diagonals (in* $\mathtt{ENV}^{(r)}$*), whereas the solid lines represent legal diagonal and vertical segments.*

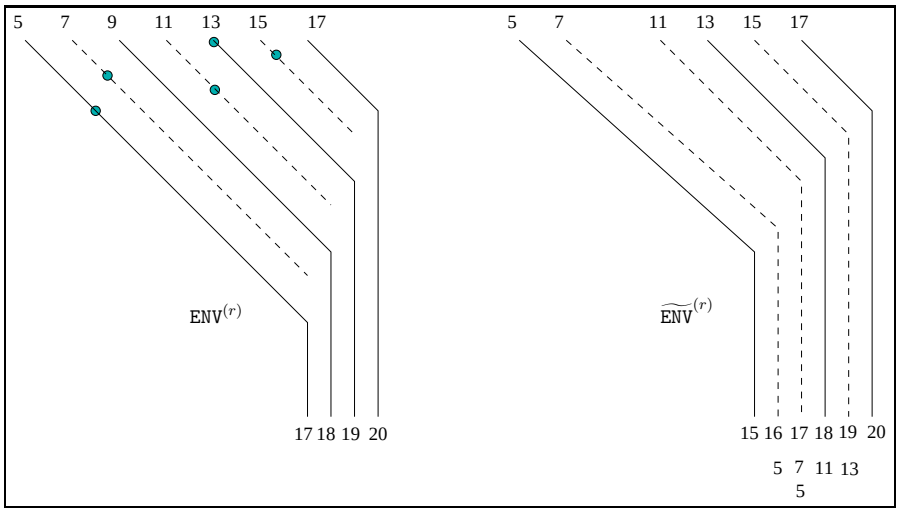
(N.B.: Since the queue is empty, this means that there is no query on the legal diagonal that corresponds to $s_{i-1}-2$, whereas e_{i-1} as matched to s_{i-1} (see Case 2).)

Note that, in both cases, we matched the vertical line ending at e_i to the very legal diagonal to which it is connected in $\mathtt{ENV}^{(r)}$.

2. The following two cases corresponds to the presence of a query on the relevant diagonal (determined by the case).
   (a) If s_{i-1} corresponds to a legal diagonal and there is a query on the illegal diagonal corresponding to $s_{i-1}-2$, then we set $s_i = s_{i-1}-2$. If there is also a query on the legal diagonal corresponding to $s_i - 2$, then we add $s_i - 2$ to the queue.
   (b) Similarly, if s_{i-1} corresponds to an illegal diagonal and there is a query on the illegal diagonal corresponding to $s_{i-1}-4$, then we set $s_i = s_{i-1}-4$, and add $s_i - 2$ to the queue if there is a query on the diagonal that correspond to it.

(Recall: The foregoing four cases are depicted in Figure 23.)

- If the queue is not empty, we set s_i to be index at the front of the queue, and remove it from the queue. (This means that we matched e_i to s_i, which means that we certainly matched e_i differently than in $\mathtt{ENV}^{(r)}$.)



The full circles on the l.h.s. image correspond to queries and the state of the queue is depicted on the bottom right-hand side. In particular, when column 19 is matched with diagonal 15, column 13 is added to the queue (which was previously empty) since it ends in $\texttt{ENV}^{(r)}$ in column 13 and it was queried. When it is removed from the queue to be matched with column 18, diagonal 11 is added. When diagonal 11 is removed from the queue to be matched with column 17, diagonals 7 and 5 are added to the queue. When diagonal 7 is removed from the queue, it is matched with column 16, and when diagonal 5 is removed from the queue, it is matched with column 15.

Figure 24: *Detail for the proof of Claim 6.8.4. An illustration for the way the matching is constructed when the queue is non-empty.*

If there is a diagonal starting in position $(0, s)$ such that $s < s_i$ and in $\texttt{ENV}^{(r)}$ this diagonal ends in column e_i, and if there is a query (in Q) on this diagonal, then s is added to the end of the queue. If there are two such indices (one corresponding to a legal diagonal and one to an illegal one), then they are both added (where the illegal one, which has the larger index, is added first).[37]

The overall process of setting the s_i's while handling the queue is depicted in Figure 24.

Once we exit the illegal region and enter the first legal region and as long as the queue is not empty, the matching is performed as in the last case (of a non-empty queue while being in the illegal region). The only difference is that in the current case at most one diagonal may enter the queue in each iteration; this happens when there is a query on the legal diagonal that ends in column e_i (in $\texttt{ENV}^{(r)}$). This means that the queue will become empty after at most $|Q|$ iterations that refer to the first legal region, because if the queue contained q elements, then at least q queries were made on diagonals that reside in the illegal region. Note that we may get into trouble only if $r < |Q|$, since otherwise the queue will become empty before we finish dealing with the first legal region. (Here we use the hypothesis $0.75n \notin e_r(G(D(Q) \cap I_r) \cup A_{r,|Q|})$.) Once the queue becomes empty, the matching is performed as in the second legal interval; that is, e_i is matched with $s_i = s_{i-1} - 2$.

[37]Recall that in $\texttt{ENV}^{(r)}$ there are two diagonals that end in each column in $[0.75 + r, 0.8 + r]$ (and these diagonals are in the illegal interval).

The behavior of the queue is reflected (or rather upper-bounded) by the definition of the game and the function G defined above. Specifically, the queue's size is incremented only if a queue is made on a relevant diagonal, and otherwise the queue size is decremented (unless it is empty already). The constructed $\widetilde{\texttt{ENV}}^{(r)}$ differs from $\texttt{ENV}^{(r)}$ only on columns that correspond to iterations in which the queue was not empty, and if no queries were made on these columns then $\widetilde{\texttt{ENV}}_Q^{(r)}$ and $\texttt{ENV}_Q^{(r)}$ are identical. Finally, the claim hypothesis (i.e., $e_r(G(D(Q) \cap I_r) \cup A_{r,|Q|}) \cap C(Q) = \emptyset$) implies that there are no queries on these columns (i.e., the columns that correspond to iterations in which the queue was not empty). The claim follows. ■

Combining Claims 6.8.3 and 6.8.4, it follows that, for every $Q \subseteq [[n]] \times [n]$, with probability $O(|Q|^2/n)$ over the choices of $r \in [0.1n]$, the view $\texttt{ENV}_Q^{(r)}$ is consistent with a legal environment of the moving object evolution rule. Thus, any nonadaptive one-sided error tester of query complexity $q(n)$, must accept a randomly chosen $\texttt{ENV}^{(r)}$ with probability at least $O(q(n)^2/n)$, where the probability is taken over both the choice of $r \in [0.1n]$ and the tester internal coin tosses. However, all these $\texttt{ENV}^{(r)}$'s are far from evolving according to the said rule, and thus the tester is not allowed to accept them with probability greater than 1/3. Hence, $q(n) = \Omega(\sqrt{n})$ must hold. ■

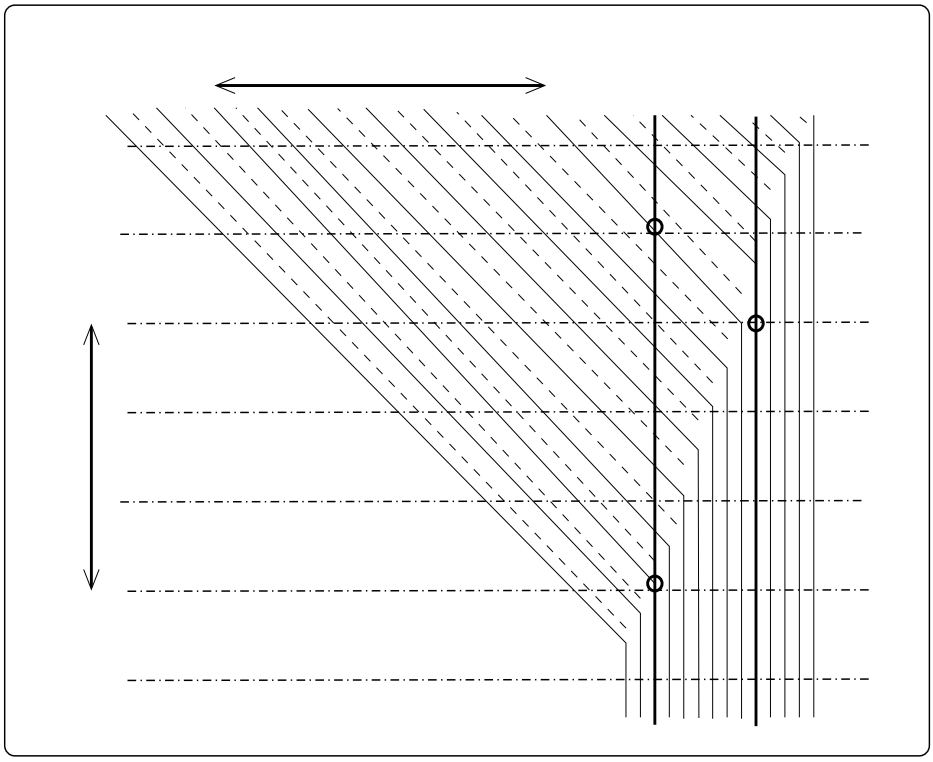


Figure 25: *An illustration for Remark 6.9. The figure shows diagonals that were included in the first set of queries and two vertical lines who's intersections with the horizontal lines were included in the second set of queries. The three small cycles indicate intersection points that (together with the first set of queries) provide a proof of illegality of the environment: Too many diagonals (specifically, all diagonals prior to i and subsequent to j) stop between these two verticals (which can fit only 2/3 of that amount).*

Remark 6.9 *The construction used in the proof of Theorem 6.8 cannot prove a stronger lower bound, since a nonadaptive machine making $O(\sqrt{n})$ queries may find evidence for its inconsistency with the evolution rule. Consider a tester that, for every $\ell \in L \stackrel{\text{def}}{=} \{0.40n, 0.41n, ..., 0.79n\}$, queries $\mathtt{ENV}_0^{(r)}$ on the locations $\{\ell + 2i + 1 : i \in [10\sqrt{n}]\}$ and queries $\mathtt{ENV}^{(r)}$ on the points $\{(j \cdot \sqrt{n}, \ell + 0.1n + i \cdot \sqrt{n}) : j \in [\sqrt{n}], i \in [2]\}$. The first set of queries reveals $10\sqrt{n}$ consecutive diagonals that start in I_r* (e.g., $\{0.4n + 2r' + 2i + 1 : i \in [10\sqrt{n}]\}$, where $r' = \lceil r/0.01n \rceil \cdot 0.01n \in L$)*, whereas the second set of queries reveals that the stopping times of these $10\sqrt{n}$ diagonals are too close (i.e., they stop at distance at most $\sqrt{n}$ from two columns that are at distance $2.5\sqrt{n}$ apart, whereas in a legal evolution the distance should be $5\sqrt{n}$). See Figure 25.*

We wonder what is the total query complexity of the best nonadaptive *one-sided error* tester for the consistency of environments of the form $\mathtt{ENV} : [n]^2 \to \{-1, 0, 1, \perp\}$ with the moving object evolution rule. More importantly, we wonder about the query complexity of general (i.e., adaptive) time-conforming *one-sided error* tester for this property.

Open Problem 6.10 *As stated above, we do not know whether the result of Theorem 6.8 extend to arbitrary* (i.e., adaptive) *time-conforming testers. That is, what is the total query complexity of the best time-conforming* one-sided error *tester for the consistency of environments of the form $\mathtt{ENV} : [n]^2 \to \{-1, 0, 1, \perp\}$ with the moving object evolution rule? In particular, is it $\Omega(n^c)$, for some $c > 0$, or is it $\mathrm{poly}(\epsilon^{-1} \log n)$, or something in-between?*

We believe that this open problem may be instructive also towards the study of testing variable movement and/or movement in multi-dimensional environments (as initiated in Section 6.2).

6.2 Variable movement in multi-dimensional environments

Here we extend the model considered in Section 6.1 in two ways. The first (and obvious) extension is from one dimension to $d \geq 2$ dimensions. The second extension is considering objects that may vary their movement according to their internal state (rather than merely stop as a result of a collision).

Fixed multi-dimensional interruptible movement. A natural question is whether the result of Theorem 6.7 can be extended to $d \geq 2$ dimensions. Actually, there are two natural models that may be considered. In the *first model*, objects stop only after a head-on collision, which happens when they are moving on the same line and in opposite directions, and otherwise they just cross each other's paths (where these two paths spans a two-dimensional space). (A collision with a standing object always counts as a head-on collision.)[38] In the *second model*, objects stops whenever their paths collide (regardless if this is a head-on collision or a side-collision).

Open Problem 6.11 *For any fixed $d > 1$, does there exists a time-conforming oracle machine of* (total) *time complexity* $\mathrm{poly}(1/\epsilon)$ *that tests the consistency of evolving environments with the first model* (i.e., the head-on collisions model) *of moving objects in d-dimensions? Ditto with respect to the second model* (i.e., the colliding paths model).

[38] Postulating the opposite does not seem reasonable because this would distinguish standing objects that originally moved along the same line from standing object that originally moved on a different line.

Testing in the first model can be reduced to testing in the second model,[39] and our initial feeling was that the first model is easier to deal with. Furthermore, it seems as if testing the evolving d-dimensional environment in the first model can be reduced to testing a small sample of the $((3^d-1)/2)\cdot n^{d-1}$ one-dimensional lines that represent possible movement paths. The intuitive (but inaccurate) justification is that stopping may occur only due to head-on collisions, whereas head-on collision occur only if the two objects move on the same one-directional line (but in opposite directions). This is correct if a collision with a standing object does not count as a head-on collision (when the standing object has originally moved along a different path), but we have postulated the opposite.

Variable (state-dependent) movement. In the models considered so far, the state of an object was identical to its direction of movement. Furthermore, the only change in state allowed was from moving in a specific direction to standing in place, and such a change took place (only) as a result of a collision with another object. Here we consider moving objects that hold a more complex state, which encodes not only their current direction of movement but also some additional information. Such objects can vary their direction of movement also when they do not collide with any other object, and their response to a collision is not necessarily stopping (although we still do not allow two objects to occupy the same location at the same time).

Unfortunately, this model of moving objects is not easier to handle than the general model of fully visible states. That is, for any $d \geq 1$, testing consistency of the evolution of d-dimensional environments with fully visible states is not made easier when we confine these environments to represent the variable (state-dependent) movement of objects in a d-dimensional grid. This follows from the fact that such (stateful) moving objects can emulate the evolution of any environment having fully visible states.[40] Details follow.

For any evolution rule $\Gamma : \Sigma^{3^d} \to \Sigma$, consider moving objects (in a d-dimensional grid) that encode (in their own state) the state of the corresponding cell. Initially, for each $\overline{i} = (i_1, ..., i_d) \in [n]^d$, the state of location $\overline{i}$ is encoded in the state of an object that resides in location $(2i_1 + 1, ..., 2i_d + 1)$. The emulation of a single evolution step (according to Γ) is done by having the object "communicate" the "encoded state" to their neighbors by their movement in the next $O(3^d \cdot \log |\Sigma|)$ time units.

For simplicity, we consider the case of $d = 1$, and envision the objects as residing on a horizontal line. The communication takes place in $8 \log_2 |\Sigma|$ time units, where in the first $4 \log_2 |\Sigma|$ time units only objects residing in locations $i \equiv 1 \pmod 4$ move. Specifically, a bit (in the encoding of the state in Σ) is communicated to both neighbors by the movement in the corresponding 4 time units

[39] We reduce testing in the first model to testing a promise problem in the second model such that only head-on collisions may occur in this promise problem. In the promise problem, the d-dimensional grid is partitions to subcubes with side-length of 3^d. The initial configurations have objects moving in direction $\overline{\delta} \in \{-1, 0, 1\}^d$ placed at distance exactly $D(\overline{\delta})$ from the center of such sub-cube along the line (in direction $\overline{\delta}$) that pass through this center, where $D : \{-1, 0, 1\}^d \to \{0, 1, ..., (3^d - 1)/2\}$ satisfies (1) $D(\overline{\delta}) = 0$ iff $\overline{\delta} = 0^d$, and (2) $D(\overline{\delta}') = D(\overline{\delta})$ iff $\overline{\delta}' \in \{\pm\overline{\delta}\}$. That is, the initial location of an object moving in direction $\overline{\delta}$ has the form $\overline{\gamma} - D(\overline{\delta}) \cdot \overline{\delta}$, where $\overline{\gamma} \in \{(3^d - 1)/2 + i \cdot 3^d : i \in \mathbb{Z}\}^d$ is the location of the center of one of the subcubes. Thus, two moving objects may collide (i.e., enter the same location at the same time) only if they move in opposite directions, and this happens when they try to enter the center of the same subcube.

[40] The emulation presented below reveals the state of the environment that the moving objects emulate. It seems that moving objects that "communicate" only via visible movement cannot emulate an environment with hidden states. Of course, if the model of moving objects allows them to sense the hidden part of the state of a neighboring object, then an emulation of arbitrary d-dimensional environments becomes possible.

such that if the bit is 1 then the object moves one step to the left, two to the right, and finally one step to the left (returning to its initial position); if the bit is 0 then the object stays in its place in all 4 time units. Once the communication is completed, each objects holds the state of all the cells that neighbor the cell that it emulates. It then determines the new state of this cell, which completes the emulation of a single step of the cellular automata.

We conclude that, in general, evolution rules that describe variable motion of statefull objects may not be easier to handle than the general evolution of environments with fully visible states. Still, one may seek natural classes of such variable motion that are easier to handle.

Open Problem 6.12 (extremely open ended): *For any $d \geq 1$, provide natural classes of evolution rules that describe variable motion of stateful objects in d dimensions such that for each rule in the class some* (or all) *of the following holds:*

1. *Testing whether the evolution of an environment of size n^d is consistent with this rule can be done in sublinear temporal query complexity; that is, complexity* $\mathrm{poly}(1/\epsilon) \cdot o(n^d)$.

   Same for complexity $\mathrm{poly}(\epsilon^{-1} \cdot \log n)$.

2. *Learning the evolution of an environment of size n^d that is consistent with this rule can be done in polynomial time and temporal query complexity $o(n^d/\log n)$.*

   Same for temporal query complexity $\mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(n^d)/t$, *say when* $t = n$.

Same when requiring the algorithms to be time-conforming.

7 Environments of Moving Objects: The Sparse Case

The model studied in Section 6 is most adequate in the case that the number of objects is comparable to the size of the environment (which is n^d). In the current section we consider the case that the number of objects, denoted m, is very small in comparison to the size of the environment; that is, $m \ll n^d$ and even $m \ll \mathrm{poly}(\epsilon) \cdot n^d$. Actually, m will denote a given upper bound on the number of objects in the environment (rather than the actual number of objects).

7.1 The model

Analogously to the study of testing bounded degree graphs (initiated in [12]) as compared to the study of testing dense graphs (initiated in [11]), we need to modify two aspects of the testing model: (1) the queries (that the testers may perform), and (2) the distance measure (between environments).

The queries. When objects are very few in comparison to the size of the environments, oracle access to $\mathtt{ENV} : [t] \times [n]^d \to \Sigma$ (only) may be quite useless. Indeed, in such a case, it is natural to allow queries that ask for the location of a specific object at a specific time. Such natural queries are modeled by an oracle, denoted $\mathtt{LOC} : [t] \times [m] \to ([n]^d \cup \{\perp\})$, such that $\mathtt{LOC}_j(i) \stackrel{\mathrm{def}}{=} \mathtt{LOC}(j,i)$ equals the location of the i^{th} object at time j, and $\perp$ indicates that the i^{th} object is not present in the environment at time j (which includes the case that it does not exist at all). We stress that there is no predetermined order between the locations of the objects (in particular, it is not

necessarily the case that $\texttt{LOC}_1(i) < \texttt{LOC}_1(i+1)$). Furthermore, we do not assume that $\texttt{LOC}_1(i) = \perp$ implies that $\texttt{LOC}_1(i+1) = \perp$.

In addition to oracle access to LOC, we also allow oracle queries to ENV, which allows to determine the state of an object (that resides in a specific location at a specific time) as well as to inspect the neighborhood of certain objects at certain times. We assume all along that ENV is consistent with LOC; that is, if $\texttt{LOC}_j(i) = \ell \in [n]^d$, then $\texttt{ENV}_j(\ell) \neq \perp$, and if $\texttt{ENV}_j(\ell) \neq \perp$, then there must exist an $i \in [m]$ such that $\texttt{LOC}_j(i) = \ell$. Furthermore, when extending the study of the simple rule of movement considered in Section 6.1, it holds that if $\texttt{LOC}_{j+1}(i) = \ell + \delta$ such that $\delta \in \{-1, 0, 1\}^d$, then $\texttt{ENV}_j(\ell) = \delta$.

Needless to say, in such a context the notion of sublinear complexity has to be modified: Here sublinear total complexity means complexity $o(tm)$, whereas sublinear temporal complexity means complexity $o(m)$. Indeed, at the very least, we seek testers of total running time $o(tm)$, but our focus should be on testers of temporal query complexity $o(m)$, which mean that for every $j \in [t]$ the tester makes $o(m)$ queries to $\texttt{LOC}_j$ and ditto to $\texttt{ENV}_j$.

The notion of *time-conforming* algorithms also needs to be revisited: Indeed, a time-conforming algorithm is restricted to make queries that are monotonically non-decreasing with respect to the time value, where this refers to both types of queries. That is, such algorithm never make a query to either $\texttt{LOC}_j$ or $\texttt{ENV}_j$ *after* making a query to either $\texttt{LOC}_{j'}$ or $\texttt{ENV}_{j'}$ such that $j < j'$.

Objects have identity. Note that the current model endows the objects with identities. That is, the location function LOC reveals that the object that resides in location $\texttt{LOC}_{t_1}(i)$ at time t_1 is the same object that resides in location $\texttt{LOC}_{t_2}(i)$ at time t_2. In contrast, the environment function ENV (which is the only source of information in the model studied in Section 6) may only reveal that these two locations were occupied at the corresponding times, but it does not reveal whether the occupant is the same object. Note that this information is not evident even in the case of fixed-speed interrupted movement in one dimension (studied in Section 6.1). In particular, this information allows to securely identify a moving object with an object that stands at a later time.

The distance. In general, the notion of distance in property testing combines a notion of distance between structures, hereafter referred to as *absolute distance*, with a normalization by the "size of the structures", which yields a notion of a *relative distance*. In the current context, when the moving objects are very few in comparison to the size of the environments, it is not natural to consider the tested structures as having size $t \cdot n^d$ and to normalize accordingly, since in this case any evolving environment with few objects will be deemed close to the empty evolving environment (which is typically consistent with any evolution rules). Instead, we view these evolving environments as having size $m \cdot t$ (representing up to m objects that exist during t units of time). The above refers to deriving a notion of relative distance from a definition of absolute distance, whereas the latter seems non-obvious here. In Section 6 things were simple: The evolving environments were represented by a single function ENV, and the absolute distance was defined as the amount of difference between such functions (i.e., number of arguments on which they differ). In this section, every evolving environment is represented by two functions, LOC and ENV, and each of these functions is determined to a great extent by the other. Thus, three natural choices (for defining absolute distance) arise.

1. *Counting the differences between function-pairs*: In this case the absolute distance between two evolving environments, represented by $(\texttt{LOC}, \texttt{ENV})$ and $(\texttt{LOC}', \texttt{ENV}')$, is defined as the sum

of $|\{(j,i) \in [t] \times [m] : \texttt{LOC}(j,i) \neq \texttt{LOC}'(j,i)\}|$ and $|\{(j,\ell) \in [t] \times [n]^d : \texttt{ENV}(j,\ell) \neq \texttt{ENV}'(j,\ell)\}|$.

2. *Counting the differences between the location functions*: In this case the absolute distance between two evolving environments, represented by their location functions $\texttt{LOC}$ and $\texttt{LOC}'$, is just $|\{(j,i) \in [t] \times [m] : \texttt{LOC}(j,i) \neq \texttt{LOC}'(j,i)\}|$. This suggestion is adequate only in case the state of an object is fully determined by its movement, as in the setting studied in Section 6.1.

   Note, however, that even within the forgoing setting, the environment function $\texttt{ENV}$ is only partially determined by the location function $\texttt{LOC}$ in the sense that, for every $(j,\ell) \in [t] \times [n]^d$, it holds that $\texttt{ENV}_j(\ell) \neq \perp$ if and only if $\texttt{LOC}_j(i) = \ell$ for some $i \in [m]$. Indeed, if the location function $\texttt{LOC}$ describes a legal evolution, then $\texttt{ENV}$ is totally determined by it. This may not be the case, in general, since we may have $\texttt{LOC}_j(i) - \texttt{LOC}_{j+1}(i) \notin \{-1,0,1\}^d$. Still, since we only consider distances between illegal evolutions $\texttt{LOC}$ and legal evolutions, each case in which $\texttt{LOC}_j(i) - \texttt{LOC}_{j+1}(i) \notin \{-1,0,1\}^d$ is counted anyhow towards the distance from a legal evolution. In general, changing the location function at a single argument yields at most two changes in the environment function. Hence, counting differences according to the location function is closely related to counting differences between function pairs.

3. *Counting the differences between the environment functions*: In this case the absolute distance between two evolving environments, represented by their environment functions $\texttt{ENV}$ and $\texttt{ENV}'$, is defined as $|\{(j,\ell) \in [t] \times [n]^d : \texttt{ENV}(j,\ell) \neq \texttt{ENV}'(j,\ell)\}|$. This means that $\texttt{ENV}$ and $\texttt{ENV}'$ are far only if for every $\texttt{LOC}$ and $\texttt{LOC}'$ that are consistent with them, the environments that are described by $(\texttt{LOC}, \texttt{ENV})$ and $(\texttt{LOC}', \texttt{ENV}')$ are far apart.

   We note that in the case of environments that evolve according to the rule of movement, the location function is determined by the environment function. However, this does not mean that if $\texttt{ENV}$ seems to fit a legal evolution, then it uniquely determines $\texttt{LOC}$. For example, we may have $\texttt{ENV}_j(\ell) = 0$ for every $j \in [t]$ and every $\ell \in \{\ell_1, ..., \ell_m\}$, describing m standing objects, but any setting of $\texttt{LOC}$ such that for every $j \in [t]$ it holds that $\{\texttt{LOC}_j(i) : i \in [m]\} = \{\ell_i : i \in [m]\}$ is consistent with this $\texttt{ENV}$, although only the settings that satisfy $\texttt{LOC}_j(1) = \texttt{LOC}_{j+1}(1)$ (for every j) represent a legal evolution.

The discrepancy between the first two measures and the third measure is reflected in our results: See the contrast between Theorems 7.1 and 7.3. The question of which measure is "right" may depend on whether the identities endowed by the location function are viewed as inherent features of the objects or as an artifact of the model that supports location queries. Specifically, if these identities are an artifact, then it is justified to consider distances according to the environment function. Indeed, ignoring the identity in this case is equivalent to saying that we only consider label-invariant properties of environments of moving objects, which is analogous to cosidering only graph properties (i.e., properties that are closed under relabeling of the graph) in the study of testing properties of graphs.

7.2 Results

Revisiting the learning algorithm described in Section 3, note that the effective number of instantaneous configurations with m objects is $\exp(m \log n)$, assuming that d is a constant. (Such a configuration is encoded by the value of the location function at a fixed time.) This fact has an immediate effect on the complexity of learning and testing such environments, but it may be possible

to do better. In particular, as in Section 6, we seek polynomial-time (time-conforming) testers of sublinear temporal query complexity.

We shall mimic the results of Section 6, showing that (1) the case of fixed-speed movement of objects in one dimension has a very efficient tester, but (2) the general case of variable movement in d dimensions is as hard as the general case of testing d-dimensional environments for consistency with respect to an arbitrary evolution rule (with fully visible states). While Part (2) holds for all distance measures we defined above, Part (1) holds only when counting the differences according to the environment function.

Starting with Part (2), we merely observe that the emulation carried out in Section 6.2 holds also in the current context, provided that the number of objects (i.e., m) is set to equal the number of locations in the emulated environment (i.e., $m = n^d$ when we emulate an environment on $[n]^d$). We stress that for such moving objects (i.e., as used in the emulation), the location function `LOC` does not provide any information that is not available by a constant number of queries to the function `ENV`. Specifically, for each $\overline{i} = (i_1, ..., i_d) \in [n]^d$, we associate a designated object indexed $\overline{i}$ (for simplicity), and this object will reside (at all times) in locations within the sub-cube (of size 3^d) that is centered at location $(2i_1 + 1, ..., 2i_d + 1)$. Note that an environment evolving on $[n]^d$ is emulated by $m = n^d$ objects that move in $[2n + 1]^d$, but the latter may be an arbitrary fraction of a larger environment of the form $[N]^d$ and the emulation holds for any $N > 2n$ (where $m = n^d$ holds always).

Turning to Part (1), we note that the tester used in Section 6.1.1 can be adapted to the current model. In fact, the most difficult check performed by this tester – the two-sided error checking of a matching between diagonal and vertical segments – is trivial in the current context (since such a matching is explicitly provided by the location function). This allows us to obtain a one-sided error tester in the current context, which (applies also to the case of $m = n$ and) stands in contrast to Theorem 6.8. The check of non-spaced standing (performed in Step 5) is also simplified by the explicit matching between diagonal and vertical segments. However, the above can be applied only for $t \geq \text{poly}(\epsilon) \cdot n$, and this restriction is inherent (see Remark 7.2).

Theorem 7.1 (testing interruptible moving objects in the sparse model, when counting differences according to the environment function): *In the model of Section 7.1, when distances are defined according to the environment function, and for $t \geq \text{poly}(\epsilon) \cdot n$, the following holds. There exists a time-conforming oracle machine of* (total) *time complexity* $\text{poly}(1/\epsilon)$ *that tests the consistency of evolving environments with the fixed-speed movement of objects in one dimension, where colliding objects stop forever. Furthermore, the tester has one-sided error probability, but is adaptive.*

The conflict between Theorem 7.1 (which obtains a one-sided error tester of complexity $\text{poly}(1/\epsilon)$) and Theorem 6.8 (which deems such tester impossible also in case $m = n$) is rooted in the fact that the model of Theorem 7.1 provides access to the location oracle `LOC`. We note that Theorem 6.8 holds also in a model in which `ENV` returns the identity of the object residing in the queried location at the queried time (as considered in Section 7.3), but `LOC` is not available.

Proof: We start by detailing the revised tester, which is obtained by adapting the tester of Section 6.1.1. Again, we start by assuming that $t \in [\epsilon n/2, n]$ (or rather that $t \in [\text{poly}(\epsilon) \cdot n, n]$), and recommend that the reader consider the case of $t = n$. We also adopt the convention by which queries that refer to locations outside of the environment's domain (i.e., $[n]$) are not made and the tester never rejects based on them (i.e., at each check, the answer is fictitiously defined as one that will not cause rejection).

It will be again more convenient to associate the initial time period with 0 rather than with 1. We use random samples $I \subseteq [m]$ (rather than $S \subseteq [n]$) and $R \subseteq [t]$, which are each of poly$(1/\epsilon)$ size, and again augment R with $\{0\}$. For simplicity, we also augment R with $\{t\}$.

- *Spaced initial configuration check*: Analogously to Step 1 in the original tester, for every $i \in I$, we check that locations $\texttt{LOC}_0(i)$ and $\texttt{LOC}_0(i)+1$ are not both occupied by moving objects. That is, for each $i \in I$, we first obtain $s \leftarrow \texttt{LOC}_0(i)$ and if $s \neq \perp$ (which indicates that the i^{th} object is present in location s of the environment at time 0), then we check that either $\texttt{ENV}_0(s) = 0$ or $\texttt{ENV}_0(s+1) \in \{0, \perp\}$.

  Let $I' \stackrel{\text{def}}{=} \{i \in I : \texttt{LOC}_0(i) \neq \perp\}$ denote the actual set of objects in I, and $s_i \stackrel{\text{def}}{=} \texttt{LOC}_0(i)$ for every $i \in I'$.

- *Individual movement check*: Analogously to Step 2, we check that there exists $r \in R$ such that the i^{th} object moves in the same direction at all times prior to r and stands in place at all times $r' \in R$ such that $r' > r$.

  Specifically, for every $i \in I'$, if $\delta \stackrel{\text{def}}{=} \texttt{ENV}_0(s_i) \neq 0$, then we let $r \in R$ be the largest integer such that $\texttt{ENV}_r(s_i + \delta \cdot r) = \delta$, and let r^+ be its successor in R, otherwise we define $r = -1$ and $r^+ = 0$. We then check that for every $r' \leq r$ in R it holds that $\texttt{LOC}_{r'}(i) = s_i + \delta \cdot r'$, whereas there exists an e_i that resides between $s_i + \delta \cdot r$ and $s_i + \delta \cdot (r^+ - 1)$ such that for every $r' > r$ in R it holds that $\texttt{LOC}_{r'}(i) = e_i$. Note that if $\delta = \texttt{ENV}_0(s_i) = 0$ then $e_i = s_i$. (Indeed, we assume that LOC and ENV are consistent with one another.)

  We also check that for every $i \in I \setminus I'$ (that is, $i \in I$ for which $\texttt{LOC}_0(i) = \perp$), it holds that $\texttt{LOC}_r(i) = \perp$ for every $r \in R$.

- *Matching movement and standing check*: As explained upfront, an analog of Step 3 is trivial here (i.e., each diagonal segment is mapped to the vertical segment that corresponds to the same object). One may view the check regarding the existence of an $e_i \in [s_i + \delta \cdot r, s_i + \delta \cdot (r^+ - 1)]$ such that $\texttt{LOC}_{r'}(i) = e_i$ for every $r' > r$ in R (where r and r^+ are also as defined above) as a check that this explicit matching is legitimate.

- *Non-crossing movement check*: Analogously to Step 4, we just need to check that the lines of the sample do not cross each other. We can actually perform this check more intuitively via the location function: For every $i, j \in I'$, we check that if $\texttt{LOC}_0(i) < \texttt{LOC}_0(j)$, then $\texttt{LOC}_r(i) < \texttt{LOC}_r(j)$ for every $r \in R$.

- *Non-spaced standing check*: Analogously to Step 5, the tester considers the intervals of movements that it has seen in prior steps, and determines the sub-intervals that must be full of standing objects. The definition of these sub-intervals is actually simpler than in the dense case since we know the exact stopping position of each sampled object (while in the dense case we only knew their approximate stopping position). The tester then samples locations in these sub-intervals (at time t) and checks that they are all occupied by standing objects. The latter check is done via queries to $\texttt{ENV}_t$.

Note that the queries to ENV performed by the tester are chosen adaptively based on the answers obtained from the LOC oracle: This happens both in the *spaced initial configuration check* and in

the *non-spaced standing check*. In particular, the queries to $\texttt{ENV}_0$ (resp., $\texttt{ENV}_t$) are made based on the answers obtained from $\texttt{LOC}_0$ (resp., $\texttt{LOC}_t$).[41]

The above tester always accepts an environment that evolves according to the fixed-movement rule. As in the proof of Theorem 6.7, we show that if the environment ENV passes the test with probability 2/3, then it is close to an environment that evolves according to the fixed-movement rule. This will be done by modifying ENV (as well as LOC) in relatively few places so to obtain a (consistent) description of a legal movement. We shall rely on the specific distance measure (which only counts differences in the environment function) only in the (second part of the) analysis of the non-spaced standing check.

From this point on, we assume that ENV (along with the consistent LOC) passes the test with probability 2/3. By the *spaced initial configuration check* and *individual movement check*, we may infer that almost all objects start in adequately spaced locations and move in an individually legitimate manner. By omitting the few exceptional lines, we obtain an environment $\texttt{ENV}'$ in which each object moves in a way that corresponds to a legal line, which consists of a (possibly empty) diagonal segment followed by a (possibly empty) vertical segment.

We now turn to the analysis of the *non-crossing movement check*: Following the argument in the proof of Theorem 6.7, we first get rid of crossing between objects that move in the same direction. Recall that the cost of eliminating these crossing is smaller than $2 \cdot \sum_{i \in L}(|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|)$, where $L = \{i \in [m] : \texttt{LOC}_0(i) \neq \perp\} = \{i_1, ..., i_{m'}\}$ denotes the set of living objects, (s_i, e_i) denotes the starting and ending positions of the i^{th} living object such that $s_{i_1} < s_{i_2} < \cdots < s_{i_{m'}}$, and π is such that $e_{\pi(i_1)} < e_{\pi(i_2)} < \cdots < e_{\pi(i_{m'})}$. On the other hand, the number of crossings (i.e., $|\{(i,j) \in L \times L : i < j \wedge e_i > e_j\}|$) is at least $\sum_{i \in L} |i - \pi^{-1}(i)| = \sum_{i \in L} |i - \pi(i)|$. The next claim, which generalizes Claim 6.5 by decoupling the parameters n and m (which were equal in Claim 6.5), shows that $\sum_{i \in L} |i - \pi(i)|$ can be upper bounded in terms of $\sum_{i \in L}(|e_i - e_{\pi(i)}| + |s_i - s_{\pi(i)}|)$. Hence, if our tester rejects with small probability, then it must be the case that the correction cost is low.[42]

Claim 7.1.1 (Claim 6.5, generalized): *Let $r_1 < r_2 < \cdots < r_m$ be real numbers such that $r_m \leq r_1 + n$ and $\pi : [m] \to [m]$ be a permutation. If $\sum_{i \in [m]} |r_i - r_{\pi(i)}| > \varepsilon mn$, then $\sum_{i \in [m]} |i - \pi(i)| > \text{poly}(\varepsilon) \cdot m^2$.*

Indeed, Claim 6.5 corresponds to the special case in which $m = n$.

Proof: We use an analogue partition of $[m]$ into buckets, $B_{j,k}$ for $j, k \in [c]$, where $c = 2/\varepsilon$, such that $i \in B_{j,k}$ if $r_i \in [(j \pm 0.5)\varepsilon n/2)$ and $r_{\pi(i)} \in [(k \pm 0.5)\varepsilon n/2)$. The contribution of $\bigcup_{j \in [c]} B_{j,j}$ to $\sum_{i \in [m]} |r_i - r_{\pi(i)}|$ is at most $\varepsilon mn/2$, since each $i \in B_{j,j}$ contributes at most $\varepsilon n/2$. It follows that there exist $j \neq k$ such that $\sum_{i \in B_{j,k}} |r_i - r_{\pi(i)}| > \varepsilon mn/2c^2 = \varepsilon^3 mn/8$, which implies that $|B_{j,k}| > \frac{\varepsilon^3 mn/8}{n} = (\varepsilon/2)^3 \cdot m$ (since $|r_i - r_{\pi(i)}| \leq n$ for every $i \in [m]$). Continuing exactly as in the proof of Claim 6.5, we infer that $\sum_{i \in B_{j,k}} |i - \pi(i)| > |B_{j,k}|^2/2$, and the current claim follows. ■

Next, we turn to crossings between objects that move in opposite direction (or between moving and standing objects). Again, we proceed exactly as in the proof of Theorem 6.7, while noting that we may use Claim 6.6 as is (indeed, Claim 6.6 refers to n points, but this is a free parameter in

[41] While this does *not* violate the time-conforming condition, one may argue that it is natural to consider a model in which adaptivity is not allowed within the same time unit (i.e., all queries made at time j must be determined by the end of time $j-1$). In order to adhere to this more restricted model, one may replace the queries to $\texttt{ENV}_0$ (resp., $\texttt{LOC}_t$) by corresponding queries to $\texttt{ENV}_1$ (resp., $\texttt{LOC}_{t-1}$).

[42] Again, we use the fact that the size of the sample required to detect a crossing is small enough so that this sample cannot distinguish ENV from $\texttt{ENV}'$, which is ϵ_1-close to it.

that claim, and in our current application we may set it to m). Hence, we obtain an environment $\texttt{ENV}''$ that is close to $\texttt{ENV}$ such that $\texttt{ENV}''$ consists of legal lines that do not cross one another.

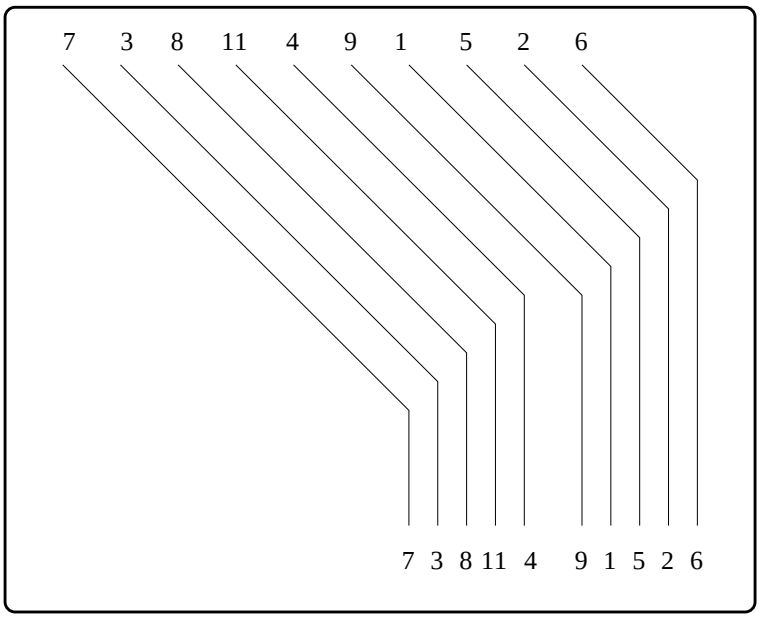


Figure 26: *Illustrating the reliance on the distance measure that counts only changes in the environment function. The objects in the figure start at a (minimal) distance 2 of one another, and (with the exception of lines 4 and 9) they end at distance 1 of one another (leaving no gaps). Note that the gap between lines 4 and 9 cannot be filled without significantly modifying half of the lines in terms of the location function.*

Finally, using the *non-spaced standing check*, we infer that, by applying a relatively small modification to $\texttt{ENV}''$, we can obtain an environment, denoted $\texttt{ENV}^{\dagger}$, in which very few vertical segments are missing within the sub-intervals that must be full of standing objects (i.e., the sub-intervals determined by the movement of the sampled objects). This is done similarly to the first stage in the analysis of Step 5 in the dense case (i.e., the proof of Theorem 6.7). In particular, we partition lines into buckets as done in the dense case. The two (related) changes are in the definition of small buckets and in the requirement regarding sampled objects in large buckets. Here we say that a bucket is *small* if the number of lines belonging to the bucket is smaller than $\epsilon_5^3 m$ (otherwise it is *large*). The cost of removing all lines belonging to small buckets is at most $\epsilon_5 mt$. We also consider partitioning the lines in each large bucket into subsets (with consecutive initial positions) of size (roughly) $\epsilon_5^3 m/4$, and assume that we sample at least one line from each such part (in each large bucket). The definition of *short* and *long* buckets remains the same and the rules for removing lines from buckets and "straightening" lines from short buckets remain unchanged. The total cost of the modifications performed is $O(\epsilon_5 mn)$ (the factor of n, rather than t is due to the straightening of short lines, as in the dense case).

We now apply the same correction procedure as in the second stage of the analysis of Step 5 in the proof of Theorem 6.7. At this point we rely on the hypothesis that distances are measured according to the environment function. The issue is that the "taking over" of a diagonal segment by a vertical segment can be charged a constant number of units (as opposed to $O(t)$ units), although

the segment moves from one object to another (i.e., from one $\texttt{LOC}(i)$ to some other $\texttt{LOC}(i')$), because we only count changes in the environment function (and do not count changes in the location function). An example where this is an issue is depicted in Figure 26. Hence, we obtain a legal environment $\texttt{ENV}^{\ddagger}$ that is close to ENV. Note that here, as in all prior steps, the distance between the corrected environment and the original one is at most $\text{poly}(\epsilon) \cdot mt$.

All the above was stated in terms of the case that $t \in [\text{poly}(\epsilon) \cdot n, n]$, and extending it to the case of $t > n$ is relatively easy. One alternative is to note that Claim 7.1.1 holds for any $t \geq n$ whereas the only other point in which the proof of Theorem 6.7 actually refers to the relation between n and t is in the analysis of the matching of diagonal and vertical segments, which we avoided here. The other alternative is just to mimic the extension presented at the end of the proof of Theorem 6.7. ■

Remark 7.2 (on the case of $t \ll \text{poly}(\epsilon) \cdot n$): *As stated above, the result of Theorem 7.1 does not extend when $t \geq \text{poly}(\epsilon) \cdot n$ does not hold. To see the problem with such a case, suppose that $m = 2n/t$ and consider an environment that consists of n/t separate sub-environments such that each sub-environment contains two objects that cross each other's path in the middle of their paths. Although such an environment is far from any environment that evolves according to the rule of movement, no oracle machine that makes $o(\min(t, \sqrt{n/t}))$ queries may see two objects in the same sub-environment (let alone a crossing). Things change if the objects indices are ordered according to their starting positions, which was* not *postulated in our model.*

As stated upfront, the result of Theorem 7.1 relies on the "counting convention" (i.e., the definition of distance according to the environment function). In contrast, if distance is defined according to the location function, then testing requires $\Omega(m^{1/4})$ queries.

Theorem 7.3 (testing interruptible moving objects in the sparse model, when counting differences according to the location function): *In the model of Section 7.1, when distances are defined according to the location function, testing the consistency of evolving environments with the fixed-speed movement of objects in one dimension requires $\Omega(m^{1/4})$ time-conforming queries, even for constant $\epsilon > 0$.*

We stress that the claim holds even if adaptive testers of two-sided error are allowed. On the other hand, the claim is restricted to time-conforming testers.

Proof: To gain intuition, consider the example depicted in Figure 26. More generally, consider a "block" of m' objects that start moving when they are at distance two apart, and stop moving at time $\Theta(t)$ such that there is a single gap among their stopping positions. Assuming that this gap is relatively close to the middle, this evolving environment is $\Omega(1)$-far from legal, when distances are defined according to the location function. Essentially, the reason is that we must "close the gap" so that the stopping of every object will be "justified". To this end we need to either modify the stopping time and position of many objects, or we need to remove many objects. On the other hand, in order to detect the illegality of this structure, a tester needs to do at least one of the following: (1) query ENV at the location of the gap; (2) query ENV at a location and time in which an object starts standing; (3) make two LOC-queries to this block (this is a necessary condition, which may not suffice.)[43] In order to avoid two LOC-queries to the same block, we have to use many

[43] For example, making two LOC-queries to this block may yield the starting and stopping locations of two objects, and if these two objects are at different sides of the gap then the differences reveal the illegality of the structure.

such blocks. Specifically, we set $m' = \sqrt{m/3}$ so that we can have $\sqrt{m}$ such blocks, which implies that the last event does not occur unless we make $\Omega(m^{1/4})$ queries.

Turning to the actual proof, we consider two distributions of evolving environments, each containing $m' \stackrel{\text{def}}{=} \sqrt{m/3}$ blocks of m' moving objects, where each object stops at some time in $[t/3, 2t/3]$ and $t = \Omega(m^{5/4})$. Each block has length $3m'$ and is partitioned into three equal parts; the "real action" will take place in the middle part, and the other parts are used to "mask" it (via randomization). The first distribution, denoted $\mathcal{D}_1$, is generated as follows (where the objects are assigned random identities in $[m]$):

1. Select uniformly a ("base") stopping time $s \in [t/3, 2t/3]$ and have the initial configuration (at time 0) contain a standing object (i.e., a vertical line) in position $m + s + 1$ (so that an object that starts at location m and moves towards this standing object will stop at time s and location $m + s$);

2. For each block $i \in [m']$ and each $j \in [m'/2]$, the initial configuration contains a rightward moving object at location $(3(i-1)+1) \cdot m' + 2j$ (i.e., the second third of each block contains moving objects that reside at the even locations at distance two apart);

3. For each block $i \in [m']$ and each $j \in [m'/2]$, select uniformly $\sigma_{i,j} \in \{0, 1\}$, and have the initial configuration contains a rightward moving object at location $(3(i-1) + 2\sigma_{i,j}) \cdot m' + 2j$ (i.e., an object is placed either at location $3(i-1) \cdot m' + 2j$, which belongs to the first third of this block, or at location $(3(i-1) + 2) \cdot m' + 2j$, which belongs to the last third).

Hence, the total length of all blocks is $m' \cdot 3m' = m$, and the number of objects in them is $m' \cdot (m'/2 + m'/2) = m/3$. The environment is obtained by applying the rule of movement to the initial configuration selected as above. Note that the initial configuration is a legal one, since the objects are at distance at least two apart from one another. Hence, $\mathcal{D}_1$ is a distribution over environments that evolve according to the moving-objects evolution rule.

The second distribution, denoted $\mathcal{D}_2$, is obtained by the following modifications to $\mathcal{D}_1$. First, for each block $i \in [m']$, we select uniformly $j_i, k_i \in [m'/2]$, remove from the initial configuration the object that resides at location $(3(i-1)+1) \cdot m' + 2j_i$ and place it in location $(3(i-1)+1) \cdot m' + 2k_i - 1$. Note that this creates an illegal initial configuration, since the relocated object is at distance one from another (moving) object. Nevertheless, consider the environment obtained by applying the rule of movement to this initial configuration (see Figure 27). We next show that with very high probability, an environment selected according to $\mathcal{D}_2$ is $\Omega(1)$-far from a legal evolution (when distances are defined by the location function). Note that the relocated object (which was placed illegally) plays the same role as the gap in the motivational discussion.

Claim 7.3.1 *With probability $1 - o(1)$ an environment chosen according to $\mathcal{D}_2$ is $\Omega(1)$-far from any environment that evolves according to the moving-objects rule when distance is measured with respect to the location function.*

Proof: Consider selecting an environment according to $\mathcal{D}_2$. We shall say that a block i is *bad* if $j_i \in [m'/8]$ and $k_i \in [m'/4, 3m'/8]$, which implies that $k_i - j_i \geq m'/8$ and $(m'/2) - k_i \geq m'/8$. Since j_i and k_i are selected uniformly independently at random in $[m'/2]$, the probability that a block is bad is $1/16$. It follows that with probability $1 - \exp(-\Omega(m'))$ the fraction of bad blocks is at least $1/32$. Let $\texttt{LOC}$ be the location function of such an environment, and let $\texttt{LOC}'$ be the location

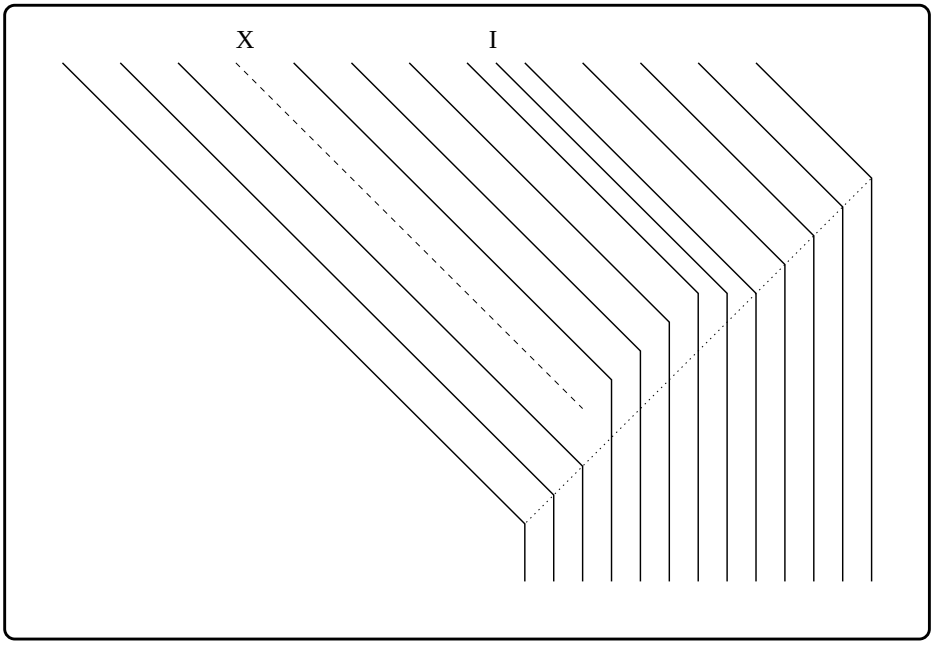


Figure 27: *Illustration of (part of) the second third of a block in the $\mathcal{D}_2$. An object was omitted from the dashed diagonal segment marked X and inserted in the diagonal segment marked I. The evolution itself is according to the rule. The dotted diagonal indicates the stopping time of moving objects in the corresponding evolution of $\mathcal{D}_1$.*

function of an environment that evolves according to the moving-objects rule. We shall show that the distance between LOC and LOC′ is $\Omega(1)$.

Observe that for any object p in LOC, if p has either a different initial location in LOC′ or a different stopping time (or it does not exist in LOC′), the difference between LOC and LOC′ due to p is $\Omega(t)$ (since in LOC the object p stops at some time $r \in [s, s+m]$, where $s \in [t/3, 2t/3]$). Consider any bad block i in LOC, and any pair of objects p and p' that belong to the second third of the block such that the initial position of p is between $(3(i-1)+1)\cdot m' + 2j_i$ and $(3(i-1)+1)\cdot m' + 2k_i - 1$ and the initial position of p' is after $(3(i-1)+1)\cdot m' + 2k_i - 1$. The main observation is that it is not possible that both p and p' will have the same initial and stopping positions in LOC and LOC′. The reason is that due to the relocation (in LOC) of the object from initial position $(3(i-1)+1)\cdot m' + 2j_i$ to initial position $(3(i-1)+1)\cdot m' + 2k_i - 1$, the difference between the starting positions of p and p' in LOC is less than twice the difference between their stopping positions. Such a behavior cannot occur in LOC′, which evolves according to the moving-objects rule.

Considering $m'/8$ disjoint pairs (p, p') as above, we infer that the contribution of a bad block to the distance between LOC and $|l'$ is $\Omega(m' \cdot t)$. The claim follows by combining this with the lower bound on the number of bad blocks in LOC. ■Theorem 7.3 follows by showing that $\Omega(m^{1/4})$ time-conforming queries are required in order to distinguish these two distributions, as we show next.

One key observation is that a time-conforming oracle machine that makes q queries is quite unlikely to make a query at the time interval $[s, s + t/100q]$. This is because queries made at the time interval $[0, s]$ are unlikely to reveal information about s, where the only exception is the

unlikely event of hitting the standing object at position $m+s+1$. Hence, if we let τ denote the time parameter of the first query made at time greater than s, then we expect τ to be greater than $s+t/3q$. Using the time-conforming condition it follows that the machine cannot make a query at the time interval $[s,\tau-1]$, even if it later learns s. However, if $t/100q > m$ (which is certainly the case when $q=o(m^{1/4})$ and $t=\Omega(m^{5/4})$), then all these later queries refer to times after all objects have stopped (i.e., to times after $s+m$). We shall show that such $o(m^{1/4})$ queries are unlikely to distinguish between the two distributions. We first give a high-level argument and then make it more rigorous.

To streamline the argument, we augment the generation process of an environment in the support of $\mathcal{D}_1$ by choices of $j_i,k_i \in [m'/2]$ (for every $i \in [m']$), which have no effect. Considering an oracle machine of query complexity $q=o(m^{1/4})$, we may assume that the following events occur, which happens with high probability: (1) no query is made at the time interval $[s,s+m]$ (in which part of the objects, but not all, stop); (2) at most one location query is made into each block; (3) for every $i \in [m']$, no query hits the diagonal that starts at location $(3(i-1)+1)\cdot m' + 2j_i$ nor the diagonal that starts at location $(3(i-1)+1)\cdot m' + 2k_i - 1$. Recall that (1) is violated with probability $O(qm/t) = o(1)$, and note that (2) is violated with probability $q^2/m' = o(1)$, whereas (3) is violated with probability $q/m' = o(1)$.

Under the above assumptions, the answers to the queries made prior to time s are identically distributed in the two distributions of evolving environments. The answers to ENV-queries made after time $s+m$ are also identically distributed in the two distributions, but the answers to the LOC-queries for time after $m+s$ (combined with the corresponding queries to $\mathtt{LOC}_0$) may reveal the stopping time of the corresponding objects. This would have allowed to distinguish the two distributions if we had not added the randomization in two thirds of each block. Specifically, the replacement of a single object in each block changes the stopping time of some objects in that block by one unit. However, this unit is masked by adding $m'/2$ random bits to it, which means that the information leakage is bounded by $O(1/\sqrt{m'})$. Hence, the total information leakage is $O(q/m^{1/4}) = o(1)$. Following is a more rigorous description of this analysis.

Claim 7.3.2 *Any time-conforming machine that makes q queries distinguishes $\mathcal{D}_1$ from $\mathcal{D}_2$ with gap $O(q/m^{1/4})$, provided that $t=\Omega(m^{5/4})$.*

Proof: In the following description, we shall construct on-the-fly an evolving environment that is distributed (approximately) according to each of the two distributions, while answering queries made to that environment. The same description applies to both distributions, except when explicitly stated otherwise. The description relies on the hypothesis that the oracle machine is time-conforming.

1. Select uniformly $s \in [t/3, 2t/3]$ and $j_i,k_i \in [m'/2]$ for every $i \in [m']$.

2. Each query that is performed at a time that resides in the interval $[0,s]$ is typically answered as it would have been answered under the first distribution. In particular, when a location query is made, we assign its index (which is in $[m]$) an initial location uniformly distributed in $[m]$, and answer it with $\perp$ (indicating that the object is not present in the environment) if the assigned location is odd. If the assigned location is even and resides in the first or last third of a block, then make a random choice (i.e., $\sigma_{i,j}$) regarding whether it exists in this third (or in the other third). The same random choices are made for relevant ENV-queries,

and in both cases the choice is recorded and the evolution is conditioned accordingly. See further details below.

Note that the distribution is indeed skewed by the fact that we never answer a location query by assigning it the standing object at location $m+s+1$; this only biases the distribution by an amount of $1/m$ (per query). More importantly, we shall abort the emulation (and output a arbitrary environment) if a query is made to the diagonals that correspond to the initial locations $(3(i-1)+1)\cdot m'+2j_i$ and $(3(i-1)+1)\cdot m'+2k_i-1$, but this event occurs with probability $O(q/m')$.

Details: We select a permutation $\pi:[m]\mapsto[m]$, representing the assignment of locations to indices of objects, and the bits $\sigma_{i,j}$, representing occupancy choices made for the first and last thirds of the blocks, on the fly. Initially, both π and the sequence of $\sigma_{i,j}$'s are fully undetermined.

When a location query $(a,b)\in\{0,1,...,t\}\times[m]$ is made, if π is undefined at b then we set $\pi(b)$ at random (among the values that do not appear yet in the range of π). Queries made to diagonals that start at odd locations are answered with an indication that no object is present there, except for the case that they refer to location $(3(i-1)+1)m'+2k_i-1$ in the i^{th} block. At this exceptional case we abort (with an arbitrary output). If a query is made to a diagonal starting at an even location at the second third of a block, then it is answered as in $\mathcal{D}_1$, except for the case that it refers to location $(3(i-1)+1)m'+2j_i$ in the i^{th} block, which also causes abort. If a query is made to the $2j^{\text{th}}$ diagonal in the first or last third of the i^{th} block and $\sigma_{i,j}$ is undefined, then we determine $\sigma_{i,j}$ at random. The query itself is then answered as in $\mathcal{D}_1$.

We stress that the difference in the handling of location and environment queries amounts to whether or not the mapping π is used and possibly further determined: Location queries refer to π, whereas environment queries do not refer to π. In both cases, the query is answered according to the starting location of the diagonal on which it resides (as determined by π in case of a location query), which may refer to the value of the relevant $\sigma_{i,j}$.

Observe that the probability of aborting is $O(q/m')$, and otherwise all queries (at times $r\in[0,s]$) are answered in exactly the same manner according to both distributions.

3. If any query is made at a time that belongs to $[s,s+m]$, then we abort the emulation. Hence, we condition the rest of the emulation by an event that occurs with probability $1-O(qm/t)$.

4. Queries made at any time $r\in[s+m+1,t]$ are answered as follows. Environment queries are answered in a straightforward manner, since the $m/3+1$ standing objects reside in the interval $[s+(2m/3)+1,s+m+1]$ and only in it. The issue is how to answer the location queries.

   We start by making all the random choices that were left undetermined by the prior steps (i.e., the random choices for the first and last third of each block, that is, the $\sigma_{i,j}$'s, as well as the remaining random assignment of objects to locations). This determines the evolution in each of the two distributions, but these evolutions differ due to the relocation between $(3(i-1)+1)\cdot m'+2j_i$ and $(3(i-1)+1)\cdot m'+2k_i-1$ in the second distribution. We answer questions in each of the two distributions accordingly.

We stress that at the current step the process is not adaptive (i.e., the random choices are not made on-the-fly in respond to queries as in Step 2). Instead, all random choices are made at the very beginning of the current step, and all queries are answered deterministically based on these choices.

It is left to analyze the deviation in the distribution of the answers that we provide for the location queries in the last step (i.e., in Step 4). Recall that if the emulation did not abort, then the diagonals that correspond to the choices j_i and k_i were never queried, and we condition the distributions on this event. We also condition the distributions on the event that location queries (made in Steps 2 and 4) hit at most one object in each block. This holds with probability $1 - O(q^2/m')$. From this point on we fix the prefix of queries and answers performed before time $s+m+1$ (where the prefix is assumed to obey the aforementioned conditioning), and we analyze the difference between the two distributions on the remaining suffix of queries and answers for any such fixed prefix (conditioned on the second event).

In each distribution, the location of an object at time t is determined by the number of objects that reside to its right in time 0, denoted ρ (i.e., the location is $s+m-\rho$). For objects that reside in the first or last third of a block, the number ρ depends only on the random choices made for this block; this holds for both distributions, and ρ is distributed identically in both cases. For objects that reside in the second third of a block, the number ρ may differ by one unit between the two distributions (due to the relocation between $(3(i-1)+1)\cdot m'+2j_i$ and $(3(i-1)+1)\cdot m'+2k_i-1$). But this difference is "masked" by the random distribution of the number of objects in the last third, which is captured by binomial distribution $B(m'')$ counting the number of heads in m'' independent coin tosses, where $m'' > m'/4$ represents the number of choices made for this block in the last step (i.e., in Step 4). Since the variation distance between the distribution $B(m'')$ and the distribution $B(m'')+1$ is $O(1/\sqrt{m''})$, the total deviation due to these answers is $O(q/\sqrt{m'/4})$.

To summarize, we showed that a time-conforming oracle machine that makes q queries can distinguish the two distributions with probability $O(q^2/m') + O(qm/t) + O(q/\sqrt{m'}) = O(q/\sqrt{m'})$, provided that $t = \Omega(m\sqrt{m'})$. The claim follows. ■

The theorem follows by combining the above two claims. Specifically, environments generated according to $\mathcal{D}_1$ have to be accepted with probability at least 2/3, whereas (by Claim 7.3.1) environments generated according to $\mathcal{D}_2$ have to be rejected with probability at least $2/3 - o(1)$. Using Claim 7.3.2, it follows that a time-conforming tester must have query complexity $\Omega(\sqrt{m'}) = \Omega(m^{1/4})$, provided that $t = \Omega(m^{5/4})$. ■

Remark 7.4 *We comment that the distributions presented in the proof of Theorem 7.3 can be distinguished by an oracle machine of query complexity $O(\log m)$ that is not time-conforming. Such a machine can find s, by a binary search along the movement of the rightmost object, which with high probability starts at a location in $[m-O(1), m-1]$. This certainly foils the analysis presented above. More importantly, a machine that is not time-conforming can first find the stopping time of an object that belongs to the second third of a block and then check the stopping times of other objects in this block. Specifically, for a random $i \in [m]$, it can first obtain the stopping time $j = \texttt{LOC}(t,i) - \texttt{LOC}(0,i)$ of the i^{th} object by making the corresponding* LOC*-queries, and then make* ENV*-queries to a small random sample of the diagonal segment $\{(j+k, \texttt{LOC}(j,i)-k) : k \in [\pm m'/10]\}$, which means querying times $j \pm (m'/10)$ after querying time t. The point is that in the first distribution all the objects of the second third of this block stop on this diagonal, whereas in the second distribution many of these objects do not stop on this diagonal* (due to the relocation)*; see Figure 27.*

This leads to

Open Problem 7.5 *What is the query complexity of testing the consistency of evolving environments with the fixed-speed movement of objects in one dimension, when distances are defined according to the location function? Same when requiring the tester to be time-conforming.*

Analogously to Open Problem 6.12, we ask

Open Problem 7.6 (extremely open ended): *Referring to the models presented in Section 7.1, provide natural classes of evolution rules that describe variable motion of relatively few stateful objects in* $[n]^d$ *such that for each rule in the class some* (or all) *of the following holds:*

1. *Testing whether the evolution of an environment with at most* m *objects is consistent with this rule can be done in sublinear temporal query complexity; that is, complexity* $\mathrm{poly}(1/\epsilon) \cdot o(m)$.
   Same for complexity $\mathrm{poly}(\epsilon^{-1} \cdot \log n)$.
2. *Learning the evolution of an environment with at most* m *objects that is consistent with this rule can be done in polynomial time and temporal query complexity* $o(m)$.
   Same for temporal query complexity $\max(1, \mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(m)/t)$.

Same when requiring the algorithms to be time-conforming.

7.3 A twist on the model: Environment oracles that return identities

Recall that the models presented in Section 7.1 provided the tester with access to two oracles, an environment oracle `ENV` and a location oracle `LOC`. As in Section 6, we assumed that the environment oracle returns a value in $\{-1, 0, 1, \perp\}$, indicating the movement direction of an object (or its non-presence). In light of the fact that the location oracle endows objects with identities, it is natural to consider also an environment oracle that indicates these identities. Specifically, we consider an environment oracle $\texttt{ENV} : [t] \times [n]^d \to [m] \times \{-1, 0, 1\} \cup \{\perp\}$ such that $\texttt{ENV}(j, \ell) = (i, \delta)$ if at time j the i^{th} object resides in location ℓ and moves in direction δ, and $\texttt{ENV}(j, \ell) = \perp$ if no object resides in location ℓ at time j.

Note that in this model all distance measures considered in Section 7.1 are closely related. Specifically, the value of `ENV` determines the value of `LOC`, and changing one value in `ENV` yields one change in `LOC`.

Open Problem 7.7 *What is the query complexity of testing the consistency of evolving environments with the fixed-speed movement of objects in one dimension, when the environment oracle returns identities of objects? Same when requiring the tester to be time-conforming.*

We conjecture that the answer is $\mathrm{poly}(\epsilon^{-1} \log n)$.

8 Directions for Further Research

One obvious question is for which local rules $\Gamma : \Sigma^{3^d} \to \Sigma$ and viewing functions $V : \Sigma \to \Sigma'$ there exists an *efficient learning* algorithm that has *sublinear temporal query complexity* (i.e., that makes $o(n^d)$ queries to each $\texttt{ENV}_j : [n]^d \to \Sigma$). Ditto for testing. These questions are actually a research

program, having numerous appealing special cases, some of them were discussed in the previous sections. In many cases, any improvement over the running time of the exhaustive search would also be interesting. On the other hand, one may argue that our notion of efficiency is too crude and that one should seek $\mathrm{poly}(n) \cdot t$-time algorithms (rather than $\mathrm{poly}(n,t)$-time algorithms).

Another general question is for which local rules $\Gamma : \Sigma^{3^d} \to \Sigma$ and viewing functions $V : \Sigma \to \Sigma'$ is it possible to *test* the environment with $o(n^d)$ queries, and furthermore with $o(n^d/t)$ queries to each $\mathtt{ENV}_j$, where here we assume that the proximity parameter (i.e., $\epsilon > 0$) is a constant. Typically, this would mean that *testing these evolving environments requires less queries than learning them.*

We note that in the context of evolving environments *nonadaptive* algorithms have a natural appeal when one may think of the probing of the environment by sensors that are placed in predetermined locations. In such a case, one may claim that relocating these sensors at time j according to the answers provided in time $j-1$ (let alone according to the answers provided in time j itself)[44] may be infeasible or undesirable. But extending this argument may mean that it is undesirable to relocating these sensors at all, or that relocation to a small distance is to be preferred. In the case of environments of moving objects (studied in Sections 6 and 7) one may think of attaching sensors to the objects. All these possibilities are natural ramifications that beg further research.

Finally, we note that our formulations of the learning and testing tasks followed the standard definitions that refer to worst-case complexity. With respect to learning, it makes sense to consider *average-case complexity* versions; for example, the case that the initial global state (i.e., $\mathtt{ENV}_1$) is uniformly distributed in Σ^{n^d}. Average-case complexity may also be applied to the testing task, but one must be very careful regarding the choice of distributions (cf. [7], which refers to property testing at large).

Acknowledgments

We were inspired by a short presentation of Bernard Chazelle in the Property Testing Workshop that took place in January 2010 at Tsinghua University (Beijing).[45] Specifically, Bernard suggested attempting to provide a sublinear time analysis of dynamic systems, which may consist of selecting few objects and tracing their movement in time. This suggestion sounded very appealing to us, and it was the trigger for the model presented here.

We are grateful to Benny Applebaum for collaboration in early stages of this research. We wish to thank a few anonymous readers for their comments. Special thanks to the reviewers of the current manuscript (Behemoth).

This research was partially supported by the Israel Science Foundation (grant No. 671/13).

[44] Indeed, our notion of adaptive (time-conforming) algorithms allows to determine queries based on the answers obtained in the same time period. Hence, we view the time period is long enough to allow for such an adaptive probing. A natural restriction may allow for adaptivity only towards later time periods; that is, at the end of each time period, we must determine (possibly based on the answers obtained at this time period) all queries to be made in the next time period.

[45] A related collection of extended abstracts and surveys has appeared as [8].

References

[1] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, Vol. 30 (6), pages 1842–1862, 2000.

[2] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, Vol. 36 (4), pages 889–974, 2006. Extended abstract in *36th STOC*, 2004.

[3] E. Blais, J. Brody, and K. Matulef. Property Testing Lower Bounds via Communication Complexity. *Computational Complexity*, Vol. 21 (2), pages 311–358, 2012.

[4] B. Chor, J. Friedman, O. Goldreich, J. Hastad, S. Rudich, and R. Smolensky. The Bit Extraction Problem or t-Resilient Functions. In *26th IEEE Symposium on Foundations of Computer Science*, pages 396–407, 1985.

[5] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17 (2), pages 230–261, 1988.

[6] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. *SIAM Journal on Computing*, Vol. 36 (4), pages 975–1024, 2006. Extended abstract in *45th FOCS*, 2004.

[7] O. Goldreich. On the Average-Case Complexity of Property Testing. ECCC, TR07-057, 2007.

[8] O. Goldreich, editor. *Property Testing – Current Research and Surveys*. Lecture Notes in Computer Science, Vol. 6390, Springer, 2010.

[9] O. Goldreich. Short Locally Testable Codes and Proofs: A Survey in Two Parts. In [8].

[10] O. Goldreich. On the Communication Complexity Methodology for proving Lower Bounds on the Query Complexity of Property Testing. *ECCC*, TR13-073, 2013.

[11] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998. Extended abstract in *37th FOCS*, 1996.

[12] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[13] O. Goldreich and D. Ron. On Proximity Oblivious Testing. *SIAM Journal on Computing*, Vol. 40, No. 2, pages 534–566, 2011. Extended abstract in *41st STOC*, 2009.

[14] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost linear length. *JACM*, Vol. 53, No. 4, July 2006, pp. 558–655.

[15] T. Gur and R. Rothblum. Non-Interactive Proofs of Proximity. *ECCC*, TR13-078, 2013.

[16] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM J. on Disc. Math.*, Vol. 5 (4), pages 545–557, 1992.

[17] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd ACM Symposium on the Theory of Computing*, pages 80–86, 2000.

[18] M.J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory.* MIT Press, 1994.

[19] S. Raskhodnikova and A. Smith. A note on adaptivity in testing properties of bounded-degree graphs. *ECCC*, TR06-089, 2006.

[20] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, Vol. 1 (3), pages 307–402, 2008.

[21] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in TCS*, Vol. 5 (2), pages 73–205, 2009.

[22] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2), pages 252–271, 1996.

[23] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, Vol. 27/11, pages 1134–1142, 1984.

Appendix: Some Tedious Details

A.1 Some linear-time computations by one-dimensional cellular automata

In this section we show how to implement a few basic operations by a linear number of steps of a one-dimensional automaton.[46] These implementation are used for proving Theorem 4.1 (as stated, rather than for $t = \text{poly}(n)$). Recall that n denotes the number of cells in the automaton. Our description uses the transmission of tokens on the automaton, which is readily implemented by special parts of the states.

Synchronizing two endpoints of an interval. Suppose that two endpoints of an interval of $[n]$ are marked (by parts of their states), and that one of its endpoints wishes to synchronize its actions with the other endpoint. This can be done by having the initiating endpoint send two tokens to the other endpoint such that one token proceed in half the speed of the other, and having the other endpoint bounce back the faster token. Synchronization is thus achieved when the faster token reaches the initiator and the slower token reaches the other endpoint.

Computing an approximation of n. We find such an approximation by repeated bisections. In the first iteration, the two endpoints of the automaton send tokens to one another, and the meeting point of these tokens is marked as the middle point, thus partitioning $[n]$ into two intervals, each of length $\lfloor n/2 \rfloor$. The endpoints of these smaller intervals proceed in a similar fashion, and this is repeated until the intervals reach length that approximately equals the number of iterations performs. The latter condition is detected by having each endpoint maintain a unary counter (via the cells to its right) of the number of iterations performed so far. The counter value is also passed to the middle point whenever it is created. Thus, the i^{th} iteration can be implemented in $O(i \cdot n/2^i)$ steps, and the process halts when $i = \Theta(n/2^i)$. At this time, n as well as simple functions of n that are bounded by a polynomial (like $\sqrt{n}$) can be approximated in $\text{poly}(\log n)$ steps, by emulating a computation of a Turing machine, where the result of the computation is stored in the first $O(\log n)$ cells.

Furthermore, numbers such as $n' = n^c$ (for some $c \in (0,1)$), which are stored in the first $\log_2 n'$ cells, can be transformed in $\widetilde{O}(n')$-time into unary notation, where the result is stored in the first n' cells. This is done by iterations, each lasting $\text{poly}(\log n)$ steps, such that in each iteration a binary counter is decreased and a new token is sent towards the right hand side. Each token stops at the first cell that contains no such token at the relevant time.

Copying a block of n' bits to the adjacent n'-bit long block. Suppose that the adjacent block is on the right hand side of the old block. Then, we start by marking the right endpoint of the new block. This can be done by synchronizing the endpoints of the old block, and having both endpoint send token towards the end of the new block such that the token send by the right endpoint proceeds in half the speed. The meeting point of these tokens is marked as the end of the new block. Next, each cell of the old block sends a token that contains its state to the right, but does so only one time unit after its right neighbor has done so (when the rightmost cell starts this processes). When a token arrives at the right endpoint of the new block, its contents is recorded, and it disappears. Other cells record and eliminate tokens that arrive to them only after their right

[46] We are quite certain that these implementations are at least implicit in the vast literature on cellular automata.

neighbor has done so, and otherwise they forward the token to that neighbor. The entire process takes $O(n')$ steps.

Maintaining synchronized clocks. Disjoint ℓ-cell blocks can maintain synchronized clocks that count modulo 2^ℓ by first determining the distance (modulo 2^ℓ) between them, and then sending a token from one interval to the other (and off-setting the clock by the said distance). For $\ell = O(\log n)$, this means maintaining synchronized (integer) clocks for poly(n) units of time. Note that the cells within each internal can be easily synchronized up to a deviation of ℓ units of time.

A.2 Modeling moving objects via cellular automata

In this section we outline how the environments studied in Section 6 can be captured by a d-dimensional cellular automaton. In particular, each step of the model presented in Section 6 will be emulated by two steps of the cellular automaton. Recall that in Section 6 we considered objects that move at the same fixed speed in one of a few directions, as long as their paths do not cross. When their paths do cross, the objects just stop at their current place (and remain there forever). As in the beginning of Section 6.2, we actually generalize this model to movement in $d \geq 1$ dimensions.

In our model, states will encode the existence or absence of an object in the location as well as auxiliary information. In case an object is present (in this location), the state also indicates the direction in which the object "wishes" to move (i.e., the object may either want to stay in place or move to one out of the $3^d - 1$ neighboring grid points). In case no object is present (i.e., the location is vacant), the state also encodes whether or not permission is granted to some neighboring object to move to this location (and if so then this permission also points to the neighbor to which the permission is granted). A location is null if it contains no object (i.e., is vacant) and no permission is granted.

It is natural to postulate that only the existence or absence of an object in a location is visible (and that the wish or permission information is hidden from the observer). An alternative formulation that is closer in spirit to the description in Section 6, includes also the past movement direction in the state and allows this part to be visible too. For the sake of simplicity, we use the first alternative in the rest of this section, while noting that the past direction of movement can easily be deduced by probing all 3^d locations in the prior time unit.

We now turn to a description of the local evolution rule that corresponds to the model outlined above. Recall that the local rule determines the next state of a location in the environment based on the states of 3^d locations in the preceding time unit (i.e., the state of the location itself and the state of its neighbors). In the current case, the local rule postulates the following:

1. If the current location *grants permission to some direction and there exists an object in the corresponding position that wishes to move to the current location*, then the object moves to the current location (and indicates that it wishes to continue moving in that direction).

   (That is, if the central (i.e., location $\overline{0} = (0, ..., 0)$) state encodes a vacancy and permission in direction $\overline{\delta} \in \{-1, 0, 1\}^d$, and the state in location $\overline{\delta}$ encodes an object wishing to move in direction $\overline{1} - \overline{\delta}$, then the new state encodes an object wishing to move in direction $\overline{1} - \overline{\delta}$.)

2. If the current location *contains an object that wishes to move in some direction and the corresponding position indicates that permission is granted in that direction*, then the current location becomes null (i.e., encodes vacancy with no permission).

(Indeed, the combination of these two sub-rules enables the movement of an object, by making sure that the object appears in the new location and disappears from the old one. Furthermore, the "directed permission" guarantees that a single object moves to the currently vacant location.)

3. If the current location is *null* (i.e., it contains no object and no permission is granted) *and there exists a single neighboring object that wishes to move to this location*, then permission is granted to this object (or rather to the direction of this object).[47] (That is, the new state encodes a permission in that direction, but the object may only move to it in the next step.) The same holds if the current location is vacant and a permission is granted in direction $\overline{\delta}$, but either there is no object in direction $\overline{\delta}$ or the object in direction $\overline{\delta}$ wishes to move elsewhere.

4. If the current location is *vacant and either no neighboring object wishes to move to it or more than one object wishes to move to it*, then this location becomes null (i.e., if it was null it remains so, and if a permission was previously granted then it is voided).

5. In all other cases, the state of the current location remains unchanged. This includes the case that the current location contains an object that wishes to stay in it, and the case that the current position is null and no neighbor wishes to move to it.

The basic model captures environments in which objects move in predetermined (or constant) directions subject to "control rules" that prevent collisions. More elaborate models may be devised for environments in which objects may change their direction of movement. Such models include an "internal control state" (ICS) per each object, and the object may change its wish (and alter its ICS) according to its current ICS and the vacancies around the object.

[47] An alternative model, which seems as natural, may postulate that if several objects wish to enter the same location, then permission is granted to one of these objects (according to a predetermined order of preference).