Electronic Colloquium on Computational Complexity, Report No. 29 (2014)

On Learning and Testing Dynamic Environments*

Oded Goldreich† Dana Ron‡

March 4, 2014

Abstract

We initiate a study of learning and testing dynamic environments, focusing on environment 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 the requirement that only a small portion of the environment is inspected in each time slot (i.e., the time period between two consecutive applications of the evolution rule).

We present some general observations, an extensive study of two special cases, two separation results, and a host of open problems. The two special cases that we study refer to linear rules of evolution and to 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 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, nonadaptivity, Locally Testable Codes, One-Way Functions,

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

†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

ISSN 1433-8092

Contents

1 Introduction

We initiate a study of sublinear algorithm for testing and learning *dynamic environments that evolve according to a local rule*. That is, the content of the environment in each location and at each time 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 elements that interact at a local level (i.e., each element 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 applications, and one may view our study as a study of sublinear algorithm 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. Indeed, such motion can also be represented as an evolution of a (three-dimensional) cellular automata.

The *sublinear aspect* of our model is reflected in the postulate that the algorithm can only probe a small portion of the environment in each time slot, although the environment evolves in time (and is thus potentially different in each time slot). 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 local rules determine 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. (By a neutral value we mean a value that does not affect the evolution of neighboring cells (e.g., zero or blank).)

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 x 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,x) 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 $j < j'$.*

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 1.6 and Section 3.2).

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 state is 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 (and there are a finite number of possibilities anyhow). 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 automata.

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 states of all locations at all times). We refer to the standard notions of property testing (cf. [GGR98, RS96, Ron10]) and PAC learning (cf. [Val84, KV94]), 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*[1] 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 $\Pr[T^{V \circ \mathtt{ENV}}(t, n, \epsilon) = 1] \geq 2/3$.*
2. *If $\mathtt{ENV}$ is ϵ-far from 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 $\mathtt{ENV}$ and $\mathtt{ENV}'$ equals the fraction of entries on which $\mathtt{ENV}$ and $\mathtt{ENV}'$ disagree* (i.e., $|\{(j, i_1, ..., i_d) \in [t] \times [n]^d : \mathtt{ENV}(j, i_1, ..., i_d) \neq \mathtt{ENV}'(j, i_1, ..., i_d)\}|/tn^d$).[2]

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

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 $V \circ \mathtt{ENV}$.*[3]

[1] 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.

[2] Equivalently, we may require that if $F : [t] \times [n]^d \to \Sigma'$ is ϵ-far from $V \circ \mathtt{ENV}'$ for any environment $\mathtt{ENV}' : [t] \times [n]^d \to \Sigma$ that evolves (from the corresponding $\mathtt{ENV}'_1$) according to Γ, then $\Pr[T^F(t, n, \epsilon) = 1] \leq 1/3$.

[3] 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 learning and testing). 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 machines makes to each $\mathtt{ENV}_j$ ($\forall j \in [t]$).*

Definitions 1.1 and 1.4 capture the time-evolving nature of the environment and our goals, and distinguish the current testing and learning problems from the standard testing and learning problems regarding various structures (including these 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). Second, the temporal query complexity refers to the number of queries made at each time slot (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 of natural phenomena, who are not only forced to be time-conforming (since they cannot inspect the past nor the future) but are restricted in the amount of inspection they can perform at any time slot.

A natural question is whether the time-conforming requirement actually restricts the power of testers. In Section 3.1 (see also Theorem 1.5) 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 pairs (Γ, V) for which the time-conforming requirement causes a subexponential increase in the query complexity of testers (i.e., an increase from $\mathrm{poly}(\log n)$ to $n^{\Omega(1)}$).

Recall that *proper learning implies testing* (cf. [GGR98, 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).[4] Thus, we present testing results 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 is not easier than learning, and this holds even if the state is fully visible (see Theorem 1.9).

1.2 A taste of our 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., $d = 1$). Recall that, throughout this paper, the evolving environments are presented 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 a subexponential increase in the query complexity of testers (i.e., an increase from $\mathrm{poly}(\log n)$ to $n^{\Omega(1)}$).

Theorem 1.5 (on the time-conforming requirement, see Theorem 3.2 for a precise statement): *There exist a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, and a viewing function $V : \Sigma \to \Sigma'$,*

[4] Recall that the argument in [GGR98, Sec. 3.1] suggests that the tester first learns a hypothesis, and then checks the hypothesis's validity by an auxiliary sample (which is uniformly distributed in the function's domain). Note 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).

such that (1) any time-conforming *tester of evolution according to* Γ *via* V *requires* $\Omega(n^c)$ *queries, but (2) there exists a* (non-time-conforming) *tester of query complexity* $\mathrm{poly}(\epsilon^{-1}\log n)$ *for this property.*

The proof of Theorem 1.5 is based on notions and ideas of Gur and Rothblum [GR13]. Specifically, we refer to their notions of general MAPs and MAPs with proof-oblivious queries, and transform the separation between them into a separation between general testers and time-conforming ones. Towards this end, we construct an evolution rule that first reveals an object to be tested, and then deletes the object and reveals a corresponding proof (for a suitable MAP). While a general tester may invoke the corresponding MAP, a time-conforming tester can be transformed into an MAP that makes proof-oblivious queries.

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

Theorem 1.6 (on the benefits of adaptivity, see Theorem 3.3 for a precise statement): *There exist a constant* $c > 0$*, an evolution rule* $\Gamma : \Sigma^3 \to \Sigma$*, and a viewing function* $V : \Sigma \to \Sigma'$*, such that (1) any* nonadaptive *tester of evolution according to* Γ *via* V *requires* $\Omega(n^c)$ *queries, but (2) there exists a* (time-conforming) *tester of query complexity* $O(\epsilon^{-1}\log n)$ *for this property.*

The proof of Theorem 1.6 is based on the observation that some separations between adaptive and nonadaptive testers that hold in the standard model can be translated to analogous results regarding testing evolving environments. Our translation requires the existence of an efficient algorithm for sampling the property that is used in the separation result (of 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.

Turning to the actual study of testing 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 1.9) and its computational complexity may be NP-Hard (cf. Theorem 2.1). 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 relates to their query complexity). The first class of evolution rules that we consider is the class of linear rules.

Theorem 1.7 (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* $\Gamma : \Sigma^{3^d} \to \Sigma$ *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* (i.e., $V(\sigma) = \sigma$ for every $\sigma \in \Sigma$)*. Furthermore, the tester is nonadaptive and has one-sided error.*

The proof of Theorem 1.7 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 $O(n^\gamma)$ location in the initial configuration. We note that our upper bound on the 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 1.8 (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 1.8 appears in Section 6.1.1. The corresponding tester makes quite a few checks, which include checking that individual objects move in 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 6.7, 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 $\Omega(\log n)$ bound for general testers (of one-sided error probability). We note that the tester used in the proof of Theorem 1.8 is actually nonadaptive.

As stated above, we show that, in general, testing evolving environments may be as hard as learning them, even when the state is fully visible. That is, in contrast to Theorems 1.7 and 1.8, there are evolution rules for which testing is not easier than learning.

Theorem 1.9 (testing may have the same query complexity as learning, see Theorem 4.3 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 *via* V *and* (proper) *learning evolution according to* Γ *via* V *have* (total) *query complexity* $\Theta(n^c)$*, where in both cases we refer to a fully visible state* (i.e., V is the identity function) *and to all sufficiently small constant values of* $\epsilon > 0$.

Theorem 1.9 is proved in Section 4, which deals with fully visible states. As in the proof of Theorem 1.6, 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. However, in the current proof, 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. We mention that testing may be infeasible also in the case of fully visible states, provided that the temporal query complexity is "significantly sublinear" (where $f(m)$ is significantly sublinear if $f(m) < m^{1-\Omega(1)}$).

Theorem 1.10 (on the computational complexity of testing with sublinear temporal query complexity, see Theorem 2.2 for a precise statement): *Assuming* $\mathcal{NP} \not\subseteq \mathcal{BPP}$*, there exists an evolution rule* $\Gamma : \Sigma^3 \to \Sigma$ *such that no probabilistic polynomial-time tester for the evolution* (of n-sized environments) *according to* Γ *and* $V_{\equiv}$ *has temporal query complexity* $n^{1-\Omega(1)}$*, where* $V_{\equiv}$ *denotes the identity mapping.*

1.3 Organization

In Section 2 we present some generic observations regarding testing and learning with respect to a general local rule Γ (viewed via a general viewing function V). It seems that going beyond these generic results, one has to restrict the class of rules. Indeed, the bulk of this work focuses on two such restrictions: (1) a restriction to linear rules (studied in Section 5), and (2) a restriction to rules that represent environments of moving objects (studied in Section 6).

Additional results are presented in Sections 2–4: In particular, in Section 3 we present the separation results reviewed above, and in Section 4 we consider the case of fully visible states.

2 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.

The simple observation (eluded 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.[5] 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 lower than that of learning, at least in some natural special cases*. We examine both questions next.

2.1 On the computational complexity of learning and testing

In general, when the state is 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 parts of the state while only the final output is visible to the observer (via the visible part of the state).

Theorem 2.1 (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 Sketch: For any NP-witness relation R and a error correcting code C of constant relative distance, consider a one-dimensional automata, captured by a rule Γ, that 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 $C(1, x)$ is far from $C(b, x')$, for every $bx' \in \{0, 1\}^{1+|x|} \setminus \{1x\}$.)[6] The

[5] 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.

[6] 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.

emulation is done using the hidden parts of the state, and only the output appears in the visible part of the state. We also let the environment repeat the output configuration indefinitely, and so for sufficiently large t this repeated output dominates the area of the evolving (t-by-n) environment, forcing the tester to relate to this (encoded) output rather than to a possible substitution of it.

Hence, if x is not in this NP-set, then an evolving environment that "shows" $C(1,x)$ (i.e., repeats it in the visible state sufficiently many times) is far from being consistent with the evolution rule Γ. On the other hand, if x is in the NP-set (defined 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 defined by R: 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 "hidden" symbol, and answers queries directed to the "repetition" stage 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 on 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 to blocks of length ℓ such that $\ell = p_1(k)$ is the largest integer satisfying $p_2(\ell) \leq n$. When wishing to decide member of $x \in \{0,1\}^k$ in the set, we emulate a computation that outputs $C(x,1)$ in each of the n/ℓ blocks, which means that $n = \mathrm{poly}(|x|)$. ■

The computational difficulty asserted in Theorem 2.1 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 state is only partially visible. Nevertheless, difficulties arise also in the case that the state is 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 the issue we state the following result, where $V_{\equiv}$ denotes the identity function.

Theorem 2.2 (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, 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 and $V_{\equiv}$ within temporal query complexity n^{1-c}, where testing is with respect to some constant value of ϵ. Furthermore, the result holds for any $t = \Omega(n)$.*

Proof Sketch: We apply the construction used in the proof of (the furthermore claim) of Theorem 2.1, 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 input 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 state is 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-slots. This is the case because the total number of blocks probed at any time-slot is at most n^{1-c}, whereas the number of blocks is n/ℓ and $p_2(\ell) \cdot n^{1-c} \ll n/\ell$.

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 $n = p_2(\ell)$. In the first $p_2(\ell)$ time slots, we

answer queries as if $(x, 0^{\ell-|x|})$ is encoded in the initial configuration in each block. Let I denote the set of blocks probed in these time-slots, and note that $|I| < n/2\ell$. Then, in the subsequent time-slots, we answer queries directed to blocks in I according to $C(0, x)$, and answer queries directed to other blocks according to $C(1, x)$. (We may assume, that $(x, 0^{\ell-|x|}) \notin R$, since otherwise we can immediately decide that x is in the NP-set.) Note that if x is in the NP-set then the answers are consistent with an evolution according to the rule Γ, and otherwise they are far from such an evolution. ■

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 next).

2.2 On testing versus learning

We observe that for almost all local rules of evolution of d-dimensional environments, Γ, learning the environment require $\Omega(n^d)$ queries (in total). On the other hand, in some natural cases (i.e., for some natural rules Γ), testing can be done using $o(n^d)$ queries (in total). (We stress that the $\Omega(n^d)$ lower bound on learning holds for these classes too.) One class of rules that supports the foregoing observation is the class of linear rules. Let us start by considering this class.

Linear rules. Here we assume that Γ is a linear function[7] (over the field Σ) and that the state is fully visible (i.e., V is the identity function). It follows that each entry in $\mathtt{ENV}$ is a linear expression in the entries of $\mathtt{ENV}_1$. Thus, testing reduces to sampling sufficiently many values, viewing each as a linear equation in the variables that correspond to the entries of $\mathtt{ENV}_1$, 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 $\mathtt{ENV}(\cdot, i_1, i_2, i_3)$ are properly related (i.e., $\mathtt{ENV}_j(i_1, i_2, i_3) = 5^{j-1} \cdot \mathtt{ENV}_1(i_1, i_2, i_3)$ 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).[8] 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 degenerated cases (i.e., when Γ is a constant function).[9] 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 testers exist for any linear Γ over a finite field F of prime order: See Theorem 1.7.

[7] In this write-up, the term *linear functions* also includes affine ones.

[8] If all checked 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 $\mathtt{ENV}_j(i_1, i_2, i_3) = 5^{j-1}\mathtt{ENV}_1(i_1, i_2, i_3)$, which means that $\mathtt{ENV}$ is ϵ-close to being consistent with the evolution of $\mathtt{ENV}_1$ according to Γ. In contrast, note that the more straightforward idea of checking $\mathtt{ENV}_{j+1}(i_1, i_2, i_3) = 5\mathtt{ENV}_j(i_1, i_2, i_3)$, 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 $\mathtt{ENV}$ may be very far from being consistent with the evolution of $\mathtt{ENV}_1$ according to Γ).

[9] See Section 5.

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. [Gol]), which may be non-trivial. We refer to the code that arises by mapping the initial global state (i.e., $\texttt{ENV}_1$) to its entire evolution over time (i.e., $\texttt{ENV}$). 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.

Environments of moving objects. 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 $\text{poly}(2^d/\epsilon)$ queries (and $\text{poly}(2^d/\epsilon)$-time computations). Essentially, the tester consists of querying $O(1/\epsilon)$ random locations in $\texttt{ENV}_1$, and querying $\text{poly}(2^d/\epsilon)$ random locations in $\texttt{ENV}$ 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 $\texttt{ENV}_1$, each possible direction $\overline{\delta} = (\delta_1, ..., \delta_d) \in \{-1, 0, 1\}^d$, and each $j \in T$, we query $\texttt{ENV}_j$ 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 $\texttt{ENV}_j(i_1 + (j-1)\delta_1, ..., i_d + (j-1)\delta_d)$ 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, then $\texttt{ENV}$ 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 inconsistence). Hence:

Theorem 2.3 (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. Then, there exists a time conforming oracle machine of* (total) *time complexity* $\text{poly}(2^d/\epsilon)$ *that tests the consistency of evolving environments with respect to $\Gamma : \Sigma^{3^d} \to \Sigma$ and the identity viewing function* (i.e., $V(\sigma) = \sigma$ for every $\sigma \in \Sigma$).

In Section 6.1 we consider a slightly more complicated model in which objects are not allowed to reside in the same location in the same time. Instead, when two objects wish to enter the same location (or cross one another), they stop at their current place forever. The analysis of this evolution rule seems much 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 $\text{poly}(1/\epsilon)$ queries: See Theorem 1.8.

An 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 that testing consistency of the evolution of d-dimensional environments with fully visible state. This is the case because such (stateful) moving objects can emulate the evolution of any environment having fully visible state.

3 Two separations

In Section 3.1, 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 3.2, 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 state is not fully visible (i.e., $V : \Sigma \to \Sigma'$ is not 1-1) and when the initial configuration satisfies some local condition (i.e., in the initial configuration each cells is in a state that belong to a set of initial states). The latter condition is captured by the following adaptation of Definition 1.2.

Definition 3.1 (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* $\mathtt{ENV}$ *is ϵ-far from 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.

This restriction on the initial configuration is quite natural. In fact we use this restriction also in our study of moving objects (presented in Section 6).[10] Note that initial configurations that are characterized as in Definition 3.1 are 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).

3.1 Time-conforming testers versus general testers

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

Theorem 3.2 (on the time-conforming requirement wrt some one-dimensional rules): *There exists a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, a viewing function $V : \Sigma \to \Sigma'$, and $\Xi \subset \Sigma$ such that the following holds:*

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

The result holds for any $t = \Omega(n)$.

[10] A cellular automaton formulation of the model studied in Section 6 may assert that when being in a state that belongs to the initial states, the object is eliminated if it neighbors any other object. This enforces the requirement that no two objects are adjacent in the initial configuration.

Needless to say, the tester in Item 1 is not time-conforming. Theorem 3.2 is proved by relying on recent results of Gur and Rothblum [GR13].

Proof: Let us first outline the high level structure of our construction of an adequate pair (Γ, V). We shall consider environments of length n, which are partitioned into three $n/3$-bit long regions. For a constant $c > 0$ to be determined later, let $k = n^c$. The first region will hold an encoding of some input $x \in \{0,1\}^k$, the second region will hold an encoding of some index $i \in [k]$, and the third region will hold an encoding of x_i. The initial configuration contains a hidden n-bit long string, and the evolution of the environment will consists of four stages, each of duration $\Theta(n)$, as depicted in Figure 1.

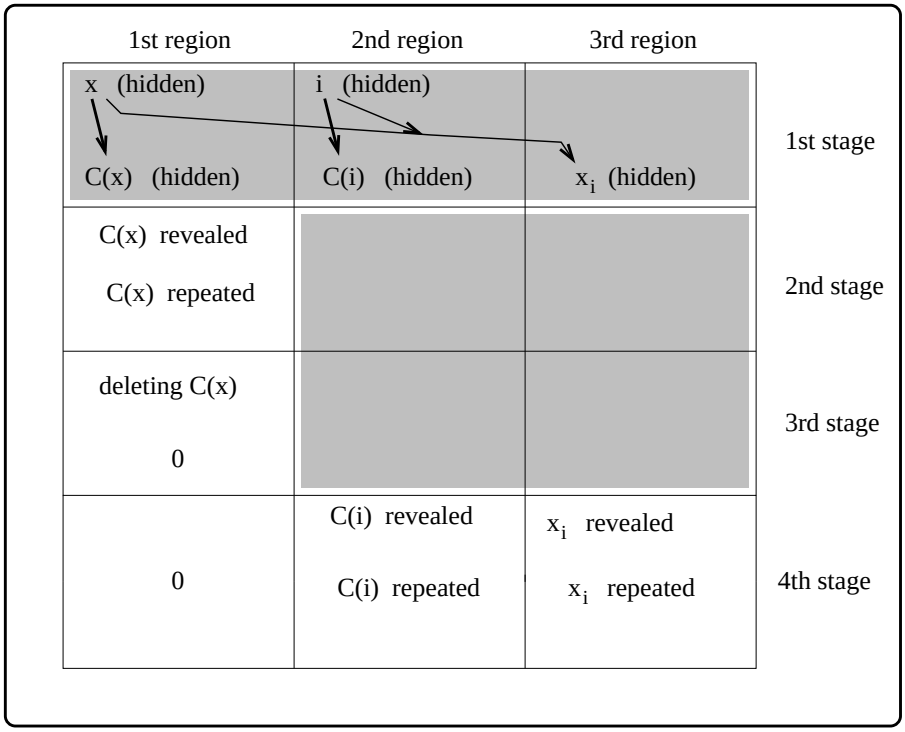


Figure 1: *The evolution of the environment of the proof of Theorem 3.2. The shaded areas represent regions and times in which the state of the corresponding cells are totally hidden.*

In the first stage, the first region is transformed into an encoding of the k-bit long string, denoted x, that is initially hidden in its first k cells; the second region is transformed into an encoding of the $\log_2 k$-bit long string, denoted i, that is initially hidden in its first $\log_2 k$ cells; and the third region is transformed into an encoding of x_i. The entire transformation as well as its end result remain hidden at this stage. 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 $t = \mathrm{poly}(n)$.)[11]

In the second stage, the contents of the first region is revealed and remains revealed for $\Theta(n)$ time slots, whereas the contents of the other regions remains hidden. In the third stage, the contents of the first region is deleted, and at the last (i.e., fourth) stage the contents of the other two regions is revealed and remains revealed for $\Theta(n)$ time slots. These three stages can be implemented by

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

a one-dimensional cellular automaton in $O(n)$ time, where the crucial observation is that we can count n time slots by having a "signal" move from one end of the environment to its other end.

We note that a general tester (which is not time-conforming) can retrieve i and x_i from the second and third regions at times that correspond to the last stage, and then recover the i^{th} bit of x from the second stage. For an appropriate choice of the encoding (i.e., a locally decodable code with rather weak parameters), these values can be recovered by making $\text{poly}(\log n)$ queries. The tester also checks that all revealed encodings are legal codewords of the corresponding codes, which for appropriate codes (i.e., locally testable codes with rather weak parameters) can be done by making $\text{poly}(\log n)$ queries. Thus, such a tester can test the consistency of the evolution with the pair (Γ, V).

In contrast, no time-conforming machine can test the consistency of the evolution with the pair (Γ, V) by making $o(k) = o(n^{1/c})$ queries. Intuitively, this is the case because by the time that i and x_i are visible, the encoding of x is no longer available. Needless to say, this intuition should be turned into an actual proof. Before getting there, let us be a bit more specific about the codes in use and the implementation of the various computations.

For starters, for any $k \in \mathbb{N}$, setting $n' = \text{poly}(k)$, we will use an error correcting code $C' : \{0,1\}^k \to \{0,1\}^{n'}$, that has constant relative distance, a polynomial-time encoding algorithm, and possesses relatively weak local testability and decodability features. In particular, we only need a (strong) codeword test (as per [GS06, Def. 3.2]) that has query complexity $\text{poly}(\epsilon^{-1} \log n')$, and a local decoder that recovers any desired bit of the information encoded in a string that is close to the code by using $\text{poly}(\log n')$ many queries (cf., e.g., [KT00]). Such codes are easy to obtain, and the standard example is a low-degree extension: For $m = (\log_2 k)/\log_2 \log_2 k$, a finite field F of size $(\log_2 k)^3$, and $H \subset F$ of size $\log_2 k$, the information viewed as $f : H^m \to F$ is encoded as a polynomial $p : F^m \to F$ of individual degree $|H| - 1$ that extends f (i.e., p agrees with f on H^m).

We cannot use C' itself as our encoding scheme, because the encoding process cannot be implemented in linear (in n') time by a one-dimensional cellular automaton. Assuming that such an implementation runs in time $n \stackrel{\text{def}}{=} k^c$, we use $(n/3n')$ repetitions of $C'(x)$ as our final encoding; that is, we let $C(x) = C'(x)^{n/3n'}$. We claim that $C(x)$ can be computed in $O(n)$ time by a one-dimensional cellular automaton with n cells, where the k bits of x are encoded in the initial states of the first k cells. The details (presented in Section A.1) include computing n, k and n', and copying n' bits to the adjacent n'-bit block in $O(n')$ steps of the automaton.[12] We can use repetitions of C' also for encoding $i \in [k]$, and can just use plain repetitions of the bit x_i, which can also be computed by the automaton in $O(n)$ steps.

As stated above, a general tester (which is not time-conforming) can test the consistency of the environment with the evolution rule Γ (and the viewing function V) by making $\text{poly}(\epsilon^{-1} \log n)$ queries. This tester uses the local testability procedure of the code C', checks various repetitions, and retrieves few bits via the local decodability procedure of C' (and of the repetition code). A crucial point here is that this tester can first retrieve i and x_i from $\texttt{ENV}_j$, where j is a random time in the fourth stage, and only later retrieve the i^{th} bit of x (which can be recovered from $C(x)$, which is visible in $\texttt{ENV}_{j'}$ for a random j' in the second stage, where $j' < j$). (Of course, this avenue is not open to a time-conforming machine.) Hence, we get

Claim 3.2.1 (fast but non-time-conforming testing): *The consistency of the environment with (Γ, V) can be tested using $\text{poly}(\epsilon^{-1} \log n)$ queries, by a machine that is not time-conforming.*

[12] 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 n'-bit block.

We next prove that any time-conforming machine that tests the consistency of the environment with the evolution rule Γ (and the viewing function V) must make $\Omega(n^c)$ queries. This proof uses ideas and notions from [GR13], which are reviewed first.

Loosely speaking, a MA proof of proximity (MAP) for a property Π with proof length $\ell : \mathbb{N} \to \mathbb{N}$ is an oracle machine M that is given auxiliary information of length $\ell = \ell(|x|)$, in addition to the input x to which it has oracle access. It is required that if $x \in \Pi$ then there exists $w \in \{0,1\}^\ell$ (which may be viewed as a short proof) such that $\Pr[M^x(w) = 1] \geq 2/3$, whereas if x is far from Π then for every $w \in \{0,1\}^\ell$ it holds that $\Pr[M^x(w) = 1] \leq 1/3$. A MAP is said to use proof-oblivious queries if its queries are independent of the auxiliary information w. In other words, first w is fixed as a function of x, then the machine gets oracle access to x, next this oracle access is disconnected and w is given to the machine, which now has to output its verdict.

Claim 3.2.2 (a reduction): *If there exists a time-conforming machine that tests the consistency of the environment with (Γ, V) while making q queries, then there exists a MAP with proofs of logarithmic length that uses $O(q)$ proof-oblivious queries for the property*

$$\Pi = \{(C(y), C(z)) : y, z \in \{0,1\}^k \ \wedge\ \exists i \in [k] \text{ s.t. } y_i = z_i = 1\}. \tag{1}$$

Furthermore, one-sided error is preserved.

Note that Π has a general MAP with proofs of logarithmic length that makes $\mathrm{poly}(\epsilon^{-1} \log n)$ queries (using i as a proof). However, Π has no MAP with proofs of logarithmic length that uses $o(k/\log n)$ *proof-oblivious* queries. This follows by combining two known results:

1. By Gur and Rothblum [GR13], a MAP with proofs of length ℓ that uses q' *proof-oblivious* queries implies a standard tester of query complexity $O(\ell q')$.

2. Using the method of [BBM12] (see also [Gol13]), any tester of the foregoing property Π must use $\Omega(k)$ queries. (This is shown by a reduction from the communication problem `set disjointness` and similar proofs appear in [GR13].)[13]

Hence, any time-conforming tester for the consistency of the environment with (Γ, V) must have query complexity $\Omega(n^c/\log n)$. So it is just left to prove Claim 3.2.2.

Proof: Given a time-conforming tester T for the consistency of the environment with (Γ, V), we construct a proof-oblivious querying MAP M as follows. For input $(C(y), C(z)) \in \Pi$ such that $y_i = z_i = 1$ for some $i \in [k]$, we shall consider the proof string i. On input (u, v) (which is supposedly in Π), the proof-oblivious querying MAP will emulate two (possibly illegal) evolutions of the environment, one that corresponds to an encoding of the triplet $(u, i, 1)$, and the other to an encoding of the triplet $(v, i, 1)$. The proof-oblivious querying MAP invokes the time-conforming tester twice, using the first (resp., second) evolution in the first (resp., second) invocation. These two invocations will be performed in parallel.

Let us consider a generic invocation that corresponds to an encoding of the triplet $(w, i, 1)$, and let t_3 denote the time slot at the end of the third stage when the deletion of the first region is completed. In this invocation w is revealed in the second stage (as the contents of the first region), and $C(i)$ and 1 are revealed in the last stage (as the contents of the second and third region, respectively). The proof-oblivious querying MAP M can emulate the queries of a time-conforming

[13] Given y and z, which represent subsets of $[k]$, the first party computes $C(y)$ and the second party computes $C(z)$. The two parties now emulate the tester of Π such that when the tester queries the i^{th} bit of $C(y)$ (resp., $C(z)$), the first (resp., second) party provides this bit to the other party.

tester T by accessing its own oracle whenever asked about the contents of the first region, and by using the proof i when asked about the contents of the second region. The crucial point is that the queries made till time-slot t_3 do not depend on the value of i, which is not visible in the environment at this period. Furthermore, the answers to queries made after time t_3 can carry no information on w, because all such information was deleted by that time. Thus, machine M can emulate all queries till time-slot t_3 by emulating the evolution of the environment on the (partially dummy) triplet $(w, 1, 1)$. Specifically, when a query is made that requires some bit in w, machine M queries the relevant part of its input-oracle.[14]

When the emulation passes beyond time-slot t_3, machine M waits for the other emulation to do so, and when both emulations pass beyond time-slot t_3, machine M stops making queries to its oracle (or is "disconnected from it" and is presented with the proof-string i). Machine M emulates all subsequent queries of T by only using the proof-string i. Since these queries refer to time-slots of index larger than t_3, they can all be answered by just using i. Specifically, when such a query is made, machine M determines whether the contents in that location is already visible, and answers accordingly (using an encoding of i or of the bit 1). Machine M accepts iff T accepts in both invocations. Thus, indeed M uses proof-oblivious queries.

If the M is indeed invokes with input $(C(y), C(z)) \in \Pi$ and given the proof-string i such that $y_i = z_i = 1$, then M will accept (with high probability or always, depending on whether T has two-sided error or one-sided error). Suppose that (u, v) is ϵ-far from Π, and that M is presented with a (false) proof i. Then, either one of the two parts is ϵ-far from being a C-codeword or they are ϵ-close to $C(y)$ and $C(z)$, respectively. In the first case, the relevant invocation of T will reject with high probability, because these codewords occupy a constant fraction of the evolution of the environment (which always produces codewords). Otherwise, either $y_i \neq 1$ or $z_i \neq 1$, and again the relevant invocation of T will reject with high probability, because the (repeated error correcting) encoding of i and 1 guarantees that the evolving environment emulated by M is far from one that is consistent with (Γ, V). ■

Recall that Claim 3.2.2 implies that any time-conforming tester for the consistency of the environment with (Γ, V) must have query complexity $\Omega(n^c/\log n)$. On the other hand, by Claim 3.2.1, the consistency of the environment with (Γ, V) can be tested using $\text{poly}(\epsilon^{-1} \log n)$ queries, by a machine that is not time-conforming. This completes the proof of the theorem (by substituting c with any constant in $(0, c)$). ■

3.2 Adaptive versus nonadaptive testers

Re-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 3.3 (on the benefits of adaptivity wrt some one-dimensional rules): *There exists a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, a viewing function $V : \Sigma \to \Sigma'$, and $\Xi \subset \Sigma$ such that the following holds:*

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

[14] Recall that w is either u or v, whereas M has oracle access to (u, v). Indeed, we assume that it is easy to determine when a bit in the first region is visible, and the emulation answers accordingly.

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 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 [RS06] that implies that *the nonadaptive query complexity of testing any non-trivial property of d-regular graphs is $\Omega(\sqrt{n})$*, where testing refers to the bounded degree model of [GR02] and a property is called non-trivial if for any n there exists an n-vertex d-regular graph that has the property and an n-vertex d-regular graph that is $\Omega(1)$-far from having the property.[15]

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 (by selecting a random $n/(d+1)$-way partition of $[n]$, which will serve as the collection of cliques, and outputting the corresponding sequence of adjacency lists). Note that this sampler generates an n'-vertex graph by using $n = O(n' \log n')$ random coins.

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 3.2. (Note that no fancy error correcting code is required here; we merely use repetitions.)

The initial configuration of the cellular automaton will encode the randomness used by the above sampler, and this initial configuration will be hidden (by use of an adequate V). The evolution of the environment will consist of two stages, each of duration $t/2$. In the first stage, the automaton will emulate the execution of the above sampler, while keeping all information hidden. In the second stage, the contents of the output of the emulation will be revealed and maintained for the rest of this stage (i.e., for $t/2 - O(n) > t/3$ time slots).

The adaptive tester just checks that (1) nothing is visible in the first stage, (2) a single string is revealed and maintained in the second stage, and (3) this string encodes a graph that has property Π_d. The checks in Items (1) and (2) are performed by simple sampling, and $O(1/\epsilon)$ samples suffice for this, while Item (3) is checked by emulating a tester for Π_d. Note that any query of the latter tester is emulated by $\log_2 n'$ queries to the string that encodes an n'-vertex graph. Hence, testing consistency of the evolving environments with (Γ, V) is reduced to testing Π_d, while increasing the query complexity by a factor of $\log n$. Using the $O(d^2/\epsilon)$-query adaptive tester for Π_d (and observing that all queries can be made to any time in $[0.667t, t]$), Part 1 follows.

To prove Part 2, we reduce the task of testing Π_d to the task of testing consistency of the evolving environments with (Γ, V), while preserving the nonadaptivity of the tester. Specifically, given a tester T for environments, we obtain a tester for Π_d by invoking T and emulating an environment that corresponds to our own input. That is, if T asks for a location that should contain a bit in the encoding of some vertex, then we query our input-oracle for the identity of this vertex, and return the adequate bit. Note that if the input to T is in Π, then the emulated evolving environment is consistent with (Γ, V), whereas if the input is ϵ-far from Π, then the emulated evolving environment is $\Omega(\epsilon)$-far from being consistent with (Γ, V). Invoking the $\Omega(\sqrt{n'})$ lower bound of [RS06], Part 2 follows. ■

[15] The result in [RS06] is more general.

4 Fully visible state

In this section we revisit the two questions raised in Section 2, while confining ourselves top environments in which the state is fully visible. That is, we consider the case in which the viewing function, V, is the identity function.

4.1 On the computational complexity of learning

We first address the question of whether the straightforward learning algorithm (presented the very beginning of Section 2) can be improved in terms of computational complexity. That is, we wish to avoid the exhaustive search (of possible values of $\texttt{ENV}_1$) that takes place in this learning algorithm.

Indeed, in the case of fully visible states, we can avoid this 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 number of queries made to each $\texttt{ENV}_j$ and the time complexity.

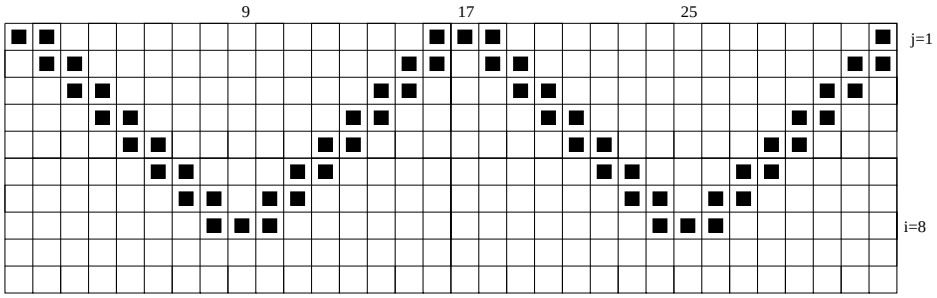


Figure 2: *The saw for $k = 16$ and $n = 32$. (Only 10 time units are shown.)*

It is instructive to start by considering the one-dimensional case. Let k be a free parameter (which governs the trade-off). In this case, rather than querying all values of $\texttt{ENV}_1$, we query $\texttt{ENV}$ at a "saw" (see Figure 2); 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 exhaustive search on each block of length k, but this is an exhaustive search on $|\Sigma^k|$ values (rather than on $|\Sigma^n|$ values).[16] Hence, this learning algorithm offers a trade-off between the number of queries made to each $\texttt{ENV}_j$ and the time complexity. (A natural problem, which is postponed for a moment, is whether we can avoid the exhaustive search on blocks of length k.)

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,

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

[16] 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 $\sigma, \tau \in \{0,1\}$.

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 $\mathtt{ENV}_j$ 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 $\mathtt{ENV}_j$. 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 $\mathtt{ENV}_j$ and the time complexity. This trade-off improves over the performance of the straightforward learning algorithm. Specifically, using $k = \sqrt[3]{\log n}$, we get a polynomial-time algorithm that makes $O(n^3/k) = o(n^3)$ queries to each $\mathtt{ENV}_j$. In general, we get:

Theorem 4.1 *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 via the identity viewing function that has temporal query complexity $O(n^d/k)$.*

We now return to the question of whether the exhaustive search on blocks of length k can be avoided. This may be possible in the one-dimensional case, but may be hard because the task at hand is at least as hard as reversing a single evolution step (i.e., from $\mathtt{ENV}_1$ to $\mathtt{ENV}_2$)), where the latter task is infeasible if one-way functions can be computed by a single step of d-dimensional cellular automata (which is quite plausible for $d \geq 2$, see [AIK10]).[17] Analogous considerations can be applied to efficient testing, but here the relevant hardness assumption seems to call for functions (computable by d-dimensional cellular automata) that have a range that is hard to recognize.

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

One-dimensional environments. In the one-dimensional case, we have $\mathtt{ENV} : [t] \times [n] \to \Sigma$ and a fixed local rule $\Gamma : \Sigma^3 \to \Sigma$. Here, there is hope to avoid the exhaustive search on blocks of length k performed above, since no one-way function can be computed by a single step of a one-dimensional cellular automaton (since a single step of such an automaton on an n-symbol long environment can be randomly reversed in time $\mathrm{poly}(|\Sigma|) \cdot n$).[18] Still, ability to randomly reverse one step does not imply ability ability to reverse several steps, and so it is not clear whether we can avoid the aforementioned exhaustive search. The question at hand is closely related to the

[17] Under reasonable assumptions, one-way functions can be computed by a single step of two-dimensional cellular automata [AIK10]. In contrast, as noted in Footnote 18, one-dimensional cellular automata cannot compute one-way functions in a single step. This does not mean that successive applications of an evolution rule can be efficiently reversed, but there seem to be hope for efficient learning (and more for testing) of one-dimensional environments.

[18] The following description follows [AIK10, Prop. 3.2]. Construct a graph with n layers, such that the i^{th} layer contains the vertex set $L_i = \{a_{-1}a_0a_1 \in \Sigma^3 : \Gamma(a_{-1}a_0a_1) = y_i\}$ and there is an directed edge from vertex $a_{-1}a_0a_1 \in V_i$ to vertex $b_{-1}b_0b_1 \in V_{i+1}$ if and only if $a_0a_1 = b_{-i}b_0$. (For $i = 1$ use $a_{-1} = 0$ and likewise for $i = 1$ and a_1.) We may add an auxiliary source vertex (at layer zero) and connect it to the vertices in L_1 that start with a zero. A valid solution corresponds to a path from this source to some vertex in L_n, and we can find it in iterations such that at the i^{th} iteration we find all vertices in L_i that are reachable from the source. We may also find the number of such paths, and so select one at random.

following: Given the result of the evolution of k steps of a one-dimensional cellular automata on an environment of length k, can we recover the initial configuration in $\exp(o(k))$-time?[19]

Open Problem 4.2 (can Theorem 4.1 be improved for $d = 1$?) *Is it the case that for every evolution rule $\Gamma : \Sigma^3 \to \Sigma$ and every $k : \mathbb{N} \to \mathbb{N}$ such that $k(n) = o(n)$, there exists an $\exp(o(k) + \log n)$-time algorithm for* (properly) *learning environments that evolve according to Γ via the identity viewing function that has temporal query complexity $o(n/k)$?*

Note that, in view of Theorem 2.2, we should not expect running time $\exp(k^{o(1)} + \log n)$.

4.2 On testing versus learning

Here we prove Theorem 1.9, which we first restate as follows.

Theorem 4.3 (testing may have the same query complexity as learning): *Let $V_{\equiv} : \Sigma \to \Sigma$ be the identity viewing function. Then, there exists 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 via $V_{\equiv}$ 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 via $V_{\equiv}$ with respect to constant ϵ has* (total) *query complexity $O(n^c)$. Moreover, the learner is non-adaptive and has temporal query complexity 1.*

Furthermore, the result holds for any $t = \Omega(n)$.

Proof: As in the proof of Theorem 3.3, 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. However, in the current proof, 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. The property that we shall use is

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

where $C : \{0,1\}^k \to \{0,1\}^{k'}$ is an explicitly constructed and good error correcting code (i.e., $k' = O(k)$ and the code has constant relative distance). For starters, we show that Π is hard to test by combining the ("communication complexity") methodology of [BBM12] with the lower bound of [CG88].

Claim 4.3.1 (warm-up): *Testing Π requires $\Omega(k)$ queries.*

[19] Note that a t-step evolution of a one-dimensional cellular automata on an environment of length n can be reversed in $\exp(\min(n, t + \log n))$-time, where the $\exp(t) \cdot n$ time-bound can be obtained by applying the procedure of Footnote 18 with Σ replaced by Σ^t. If exponentially strong one-way functions exist, then for some $t = \text{poly}(n)$ reversal cannot be performed in $\exp(o(n))$-time. But it seems unlikely that such functions can be computed in n steps of a one-dimensional cellular automata. So this leaves room for hope (regarding the case of $t = O(n)$, which seems most interesting).

Proof: Using the methodology of [BBM12] (see also [Gol13]),[20] we reduce the communication complexity problem of computing the inner product mod 2 of two k-bit long strings (which, by [CG88, Sec. 4.2], has communication complexity $\Omega(k)$) to testing Π. Consider parties A and B having inputs x and y, respectively. Then, the parties (in the joint randomness model) emulate a tester for Π by answering its queries as follows: Query $i \in [k']$ is answered by A with the i^{th} bit of $C(x)$, which is sent by A to B; query $i \in [k'+1, 2k']$ is answered by B 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. ■

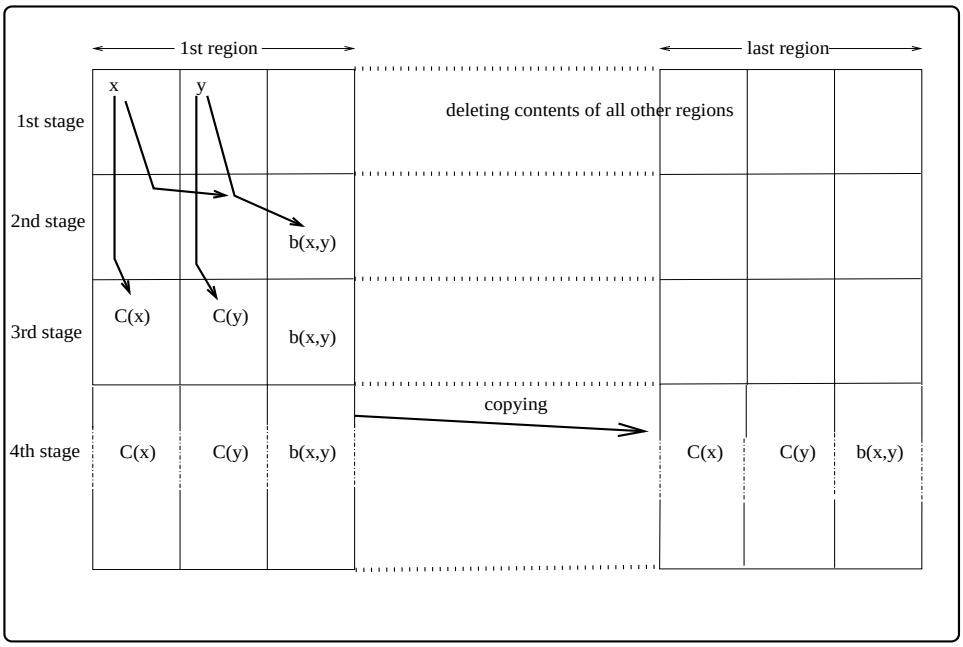


Figure 3: *The evolution of the environment of the proof of Theorem 4.3. Only the first and last regions are shown; the other regions are identical to the last region.*

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 consists of $n/3k'$ regions, each consisting of $3k'$ cells, as depicted in Figure 3.

- In the first stage, the automaton reset the states of the cells in all regions except the first one (after approximating n and determining $k = n^c$ and $k' = O(k)$ accordingly).

  This is done in order to allow the learning algorithm 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.

[20] Indeed, the actual claim is closely related to a special case of [Gol13, Thm. 4.1].

- 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 by 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 $2k$ queries, by merely 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. Note that different bits can be read (non-adaptively) at different times (within the first stage). As mentioned above, 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 follows.

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.)[21]

Claim 4.3.2 (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*

$$\Pi' \stackrel{\text{def}}{=} \left\{ \left(F(x), F(y), B(x,y)^{O(k)}, C(x)^{n^2/k'}, C(y)^{n^2/k'}, b(x,y)^{tn/4} \right) : x, y \in \{0,1\}^k \right\} \tag{3}$$

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

[21] An alternative presentation may avoid the presentation of an explicit property testing problem, and apply the methodology of [BBM12] without sticking to its specific formulation.

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 [BBM12], we reduce the communication complexity of `Disjointness` to testing Π'. Recall that in `Disjointness` the two parties are given the inputs x and y such that $I \stackrel{\text{def}}{=} \{i \in [k] : x_i = y_i = 1\}$ has cardinality at most 1, and need to decide whether or not $I = \emptyset$. Note that under the promise (that the size is at most 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 $|I| = \sum_{i \in [k]} x_i y_i$ (over the integers).) Recall that the communication complexity of `Disjointness` is $\Omega(k)$; cf. [KS92].

The reduction of the communication complexity of `Disjointness` to testing Π', proceeds as follows, where the parties A and B hold the inputs x and y, respectively. The parties emulate a tester for Π' by answering its queries such that 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 $(\Gamma, V_{\equiv})$, Part 2 follows. ■

5 Linear Rules

In this section we consider the evaluation 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

$$\Gamma(z_{-1^d}, .., z_{1^d}) = \sum_{\sigma \in \{-1,0,+1\}^d} \alpha_\sigma z_\sigma, \tag{4}$$

where the α_σ's are in F.

We shall focus on the case that the state is 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.

5.1 More on learning

As stated in Section 2.2, 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 degenerated cases (i.e., when Γ is a constant

function). This will be shown in Proposition 5.1, but before 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 sake of simplicity, we consider the case $t = n$; the argument generalizes for any $t = O(n/\epsilon)$. Let $\Gamma(z_{-1,...,-1},..,z_{1,...,1}) = \sum_{s_1,...,s_d \in \{-1,0,1\}} c_{s_1,...,s_d} z_{s_1,...,s_d} + b$, 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 of $(s_1,...,s_d) \in \{-1,1\}^d$ (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 w.r.t 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^3/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 a fully visible state), 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 $\mathtt{ENV}$, 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 $\mathtt{ENV}$.)

The above ideas may be applied also in the case that the state is 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 $V \circ \mathtt{ENV}$ is a linear combination (over the small field) of the values in the sequences in $\mathtt{ENV}_1$, where each element of $\mathtt{ENV}_1$ is viewed as a sequence of elements in the small field.

5.2 Testing is easier than learning

In this section we prove Theorem 1.7, 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 (in Section 2.2) 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. [GR11]), 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 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 ENV is consistent with Γ, then the tester accepts with probability 1. Suppose that ENV is ϵ-far from being consistent with Γ. Then, more than an ϵ fraction of the locations in ENV are inconsistent with the Γ-evolution of the first row of ENV. As we shall show (in Claim 5.3 below), the expected number of locations in the first row that determine a random location in 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 ENV is smaller than $\epsilon/2$. It follows that *the tester rejects* 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).[22]

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

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 slots that are associated with all integers);

[22] 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 $A_k \leq A_{k-1} + |D_{2^{k-1}}| \cdot A_{k-1}$, and it follows that $A_k \leq 3A_{k-1} \leq 3^k$, which implies that $A_k \leq (2^k)^{\log_2 3}$. The proof of Claim 5.2 is merely a more refined analysis of a similar nature.

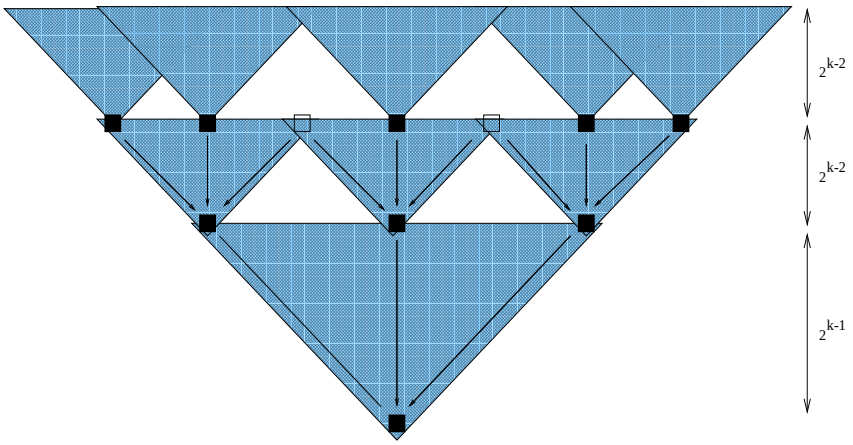


Figure 4: *Demonstrating that $A_k < A_{k-1} + 8A_{k-2}$ (for the proof of Claim 5.2). Only the shaded triangles contain locations that may influence the value of the bottom location.*

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

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

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).[23] 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 $2^{1.76} + 8 < 4^{1.76}$. ■

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 5.3 (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 5.3.1 *For $i = 1, ..., p$, and every integer ℓ, let*

$$x_\ell^{(i)} = \alpha_{-1} x_{\ell-1}^{(i-1)} + \alpha_0 x_\ell^{(i-1)} + \alpha_1 x_{\ell+1}^{(i-1)} \,. \tag{5}$$

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

Using Fact 5.3.1, 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 $x_\ell^{(i)} = T_{\ell \cdot p^{k-1}}((i-p) \cdot p^{k-1})$ (where T is as defined in the proof of Claim 5.2), and using Fact 5.3.1. We now turn to prove Fact 5.3.1.

Proof: Applying straightforward recursive substitution to Eq. (5), we get

$$\begin{aligned} x_\ell^{(p)} &= \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \cdot x^{(0)}_{\ell+\sigma_1+\cdots+\sigma_p} \\ &= \sum_{\tau \in [-p,p]} \mathtt{d}(\tau) \cdot x^{(0)}_{\ell+\tau} \qquad (6) \\ \text{where } \mathtt{d}(\tau) &\stackrel{\text{def}}{=} \sum_{\sigma_1,...,\sigma_p \in \{-1,0,+1\}:\ \Sigma_i \sigma_i = \tau} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \qquad (7) \end{aligned}$$

We start by analyzing Eq. (7). 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

[23] 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).

$\sigma \in \{\pm 1\}$, we have $\mathtt{d}(\sigma \cdot p) = \alpha_\sigma \cdots \alpha_\sigma = \alpha_\sigma^p$, 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

$$\begin{aligned} \mathtt{d}(\tau) &= \sum_{(\sigma_1,\ldots,\sigma_p)\in S_p(\tau)} \alpha_{\sigma_1} \cdots \alpha_{\sigma_p} \\ &= \sum_{j=0}^{p} |S_p(\tau, j)| \cdot \alpha_0^j \cdot \alpha_1^{(p-j+\tau)/2} \cdot \alpha_{-1}^{(p-j-\tau)/2} \end{aligned} \tag{8}$$

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. (8) 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 $\mathtt{d}(0) \equiv |S_p(0, p)| \cdot \alpha_0^p \equiv \alpha_0 \pmod p$). 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. (6), the fact follows. ■

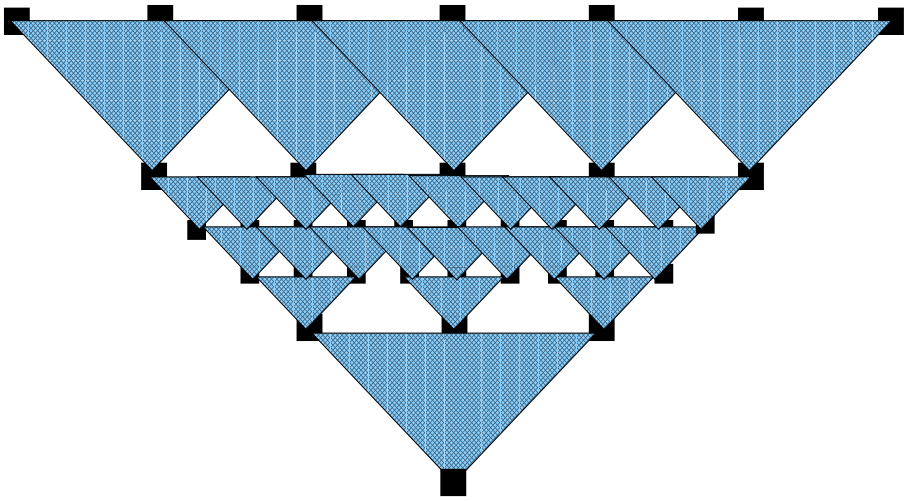


Figure 5: *The case of $p = 3$ (for the proof of Claim 5.3). Smaller triangles replace the three big triangles that would have appeared in the second level. As in Figure 4, only the shaded triangles contain locations that may influence the value of the bottom location.*

Recall that Fact 5.3.1 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 comes from there (i.e., from the second level).[24] 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 trainless 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).[25] ■

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 5.4 (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 5.4 yields an algorithm that establishes Theorem 1.7.

Proof: We generalize the proof of Claim 5.3. Here for $i = 1, ..., p$ and every d-dimensional location $v \in \mathbb{Z}^d$, we consider

$$x_v^{(i)} = \sum_{\sigma\in\{-1,0,+1\}^d} \alpha_\sigma x_{v+\sigma}^{(i-1)} \tag{9}$$

As in the proof of Fact 5.3.1, we get

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

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

[24] 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.

[25] 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 $\log_3 X \approx 1.965$.

$\mathtt{d}(\tau)$ 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 $D_{p^k} \subseteq \{-p^k,0,p^k\}^d$.

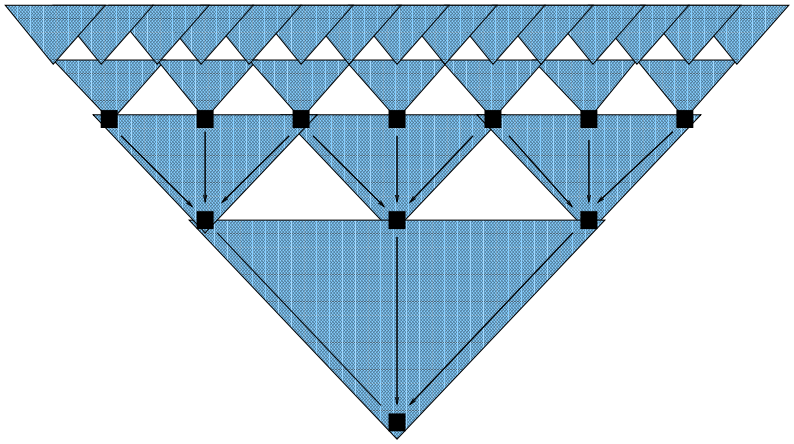


Figure 6: *A one-dimensional projection of the case of $p=2$ and $k'=3$ (for the proof of Claim 5.4). Only the shaded triangles contain locations that may influence the value of the bottom location.*

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

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. ■

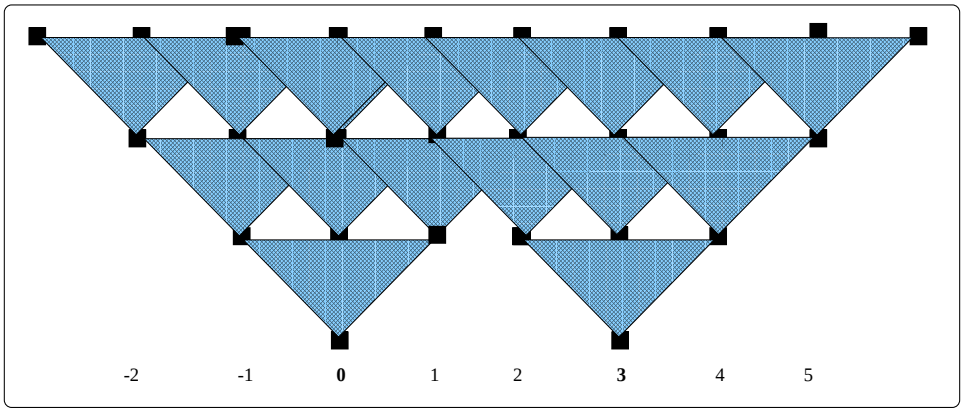


Figure 7: *A one-dimensional projection of the case of $p = 3$ and $m = 2$ (for the proof of Claim 5.4). The marked positions are scaled by 3^{k-1}. Only the shaded triangles contain locations that may influence the values of the $m = 2$ bottom locations. The bottom level represents 2^d copies of A_{k-1}, the middle level represents a copy of $A^{(6)}_{k-1}$, whereas the top level represents a copy of $A^{(8)}_{k-1}$.*

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 5.4.2 (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 $A^{(m)}_k = \sum_{i=1}^{p^k} |\cup_{j_1,...,j_d \in \{0,...,m-1\}} (\{p^k \cdot (j_1, ..., j_d)\} + D_i)|$; that is, $A^{(m)}_k$ 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, $A^{(1)}_k$ 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, $(j_1, ..., j_d) = (j'_1, ..., j'_d)$). Abusing notation such that $A^{(m)}_k$ denote a monotonic in m upper bound on $A^{(m)}_k$, the key observation is that

$$A^{(m)}_k \leq m^d \cdot A_{k-1} + \sum_{j=1}^{p-1} A^{((m-1)p+2j+1)}_{k-1} , \tag{10}$$

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. (10) is tight if we ignore cancelations that were not stated explicitly before.)[26] 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

$$A^{(m)}_k \leq m^d \cdot A_{k-1} + (p-1) \cdot A^{(N(m))}_{k-1} . \tag{11}$$

Letting $N_0(m) = m$ and $N_i(m) = N(N_{i-1}(m))$, and using Eq. (11), we get

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

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

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

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

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

where the second inequality uses $A_{k'}^{(M)} \leq M^d \cdot A_{k'}$. 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

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

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 $\sum_{i \geq 1} q^{i-1} < \frac{1}{0.9/p}$ 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 $p \geq 3$. The claim (i.e., $Q' + Q'' < 1$) follows since $(2.5/3)^d \cdot \frac{31}{27} < 1$ for every $d \geq 1$. ■

Combining Facts 5.4.1 and 5.4.2, the claim follows. ■

Conclusion. As stated above, Claim 5.4 implies the claims of Theorem 1.7, except that this was shown only in the case of $t = n$. We first observe that what Claim 5.4 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 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.

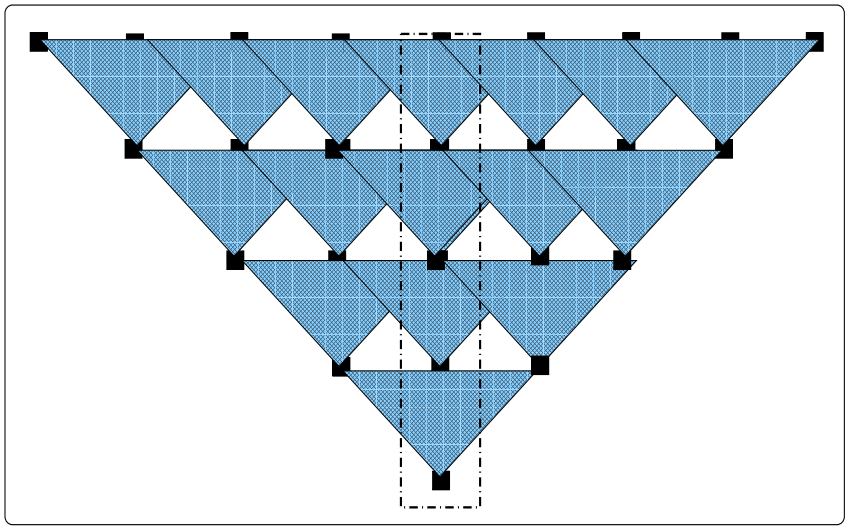


Figure 8: *A one-dimensional projection of the case of $t = 4p^\ell$ (for the analysis of the case of $t > n$). Only the shaded triangles (each of height p^ℓ) contain locations that may influence the value of the bottom location. The dashed rectangle marks the boundaries of $[t] \times [-(n-1), (n-1)]^d$.*

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).[27] 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 5.5 (Theorem 1.7, 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 the identity viewing function. Furthermore, the tester is nonadaptive and has one-sided error.*

[27] 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 5.6 *Can Theorem 5.5 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 is the following

Open Problem 5.7 (Can Theorem 5.5 be drastically improved?) *Our tester has 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. Does there exist a tester of polylogarithmic complexity?*

6 Environments of Moving Objects

In this section we consider dynamic environments that represent objects that are moving in a d-dimensional space.[28] Recall that in Theorem 2.3 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. For starters, 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 one-dimensional case is studied in Section 6.1.

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

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

[28] 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).

[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).

(We note that a single evolution step of this environment can be emulated by a pair of steps of the cellular automaton described in Section 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 have stayed 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 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.

[30] In fact, we use a small modification of that CA in which permission to enter a cell is not granted to any object when several objects wishes to enter it. (In Section A.2, in such a case, we granted entry permit to exactly one object.) The emulation 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).

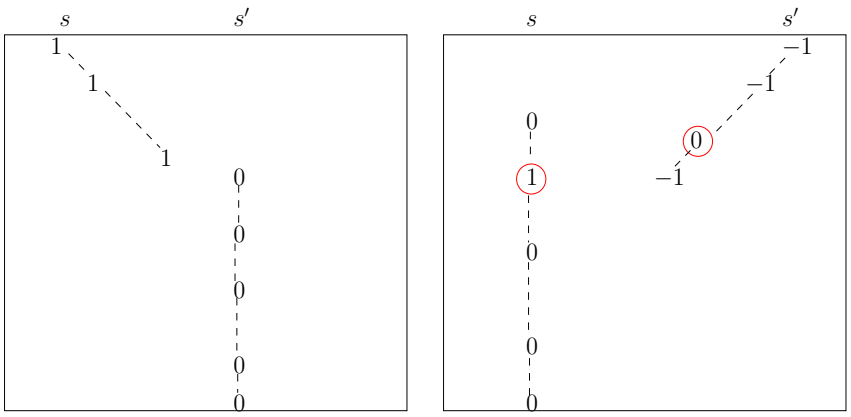


Figure 9: *An illustration for Step 2. The left image shows a legal configuration, whereas the right image shows an illegal one. The broken lines indicate the trajectories of the objects, and the violations are circled on the right.*

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.

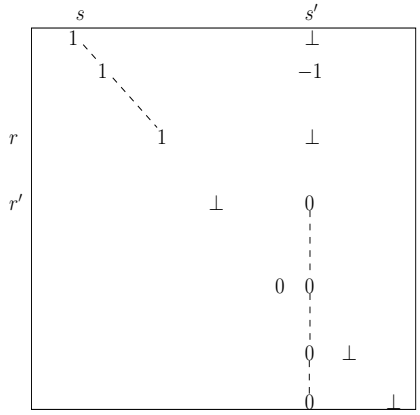


Figure 10: *An illustration for the notions of approximate stopping time and approximate time of starting to stand. The object that was in position s at time 0, stopped approximately at time r, and the object who is standing in position s' at time t, started to stand approximately at time r'.*

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.

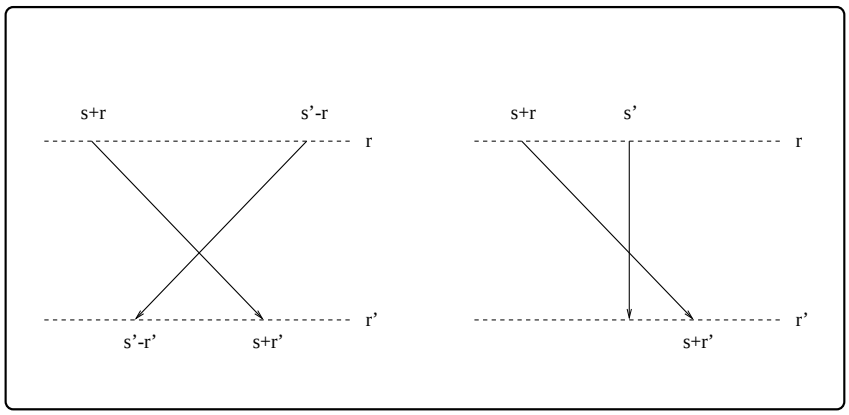


Figure 11: *An illustration for Step 4. The left image shows a crossing of two moving objects, whereas the right image shows an object moving to the right while crossing a standing object (at position s').*

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.

   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

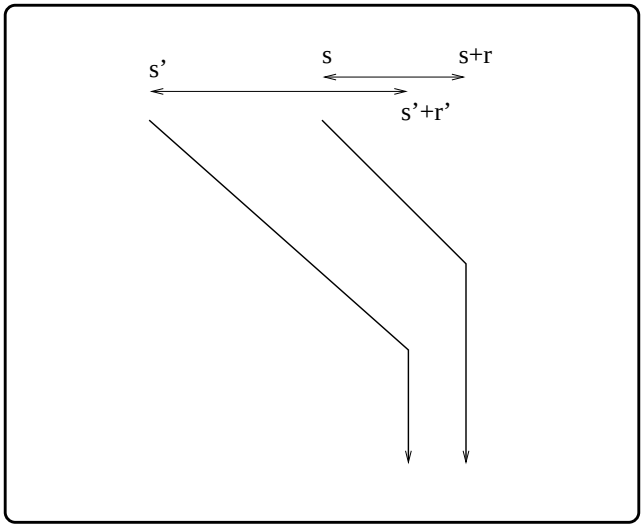


Figure 12: *An illustration for Step 5. The first object moves from position s to position $s+r$, where it stops, whereas the second objects moves from $s' < s$ to $s'+r' \in [s, s+r)$. In this case, the interval $[s'+r', s+r]$ must be filled with stopping objects (at time t).*

$r^+(s)$. 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 6.7, 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.

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 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

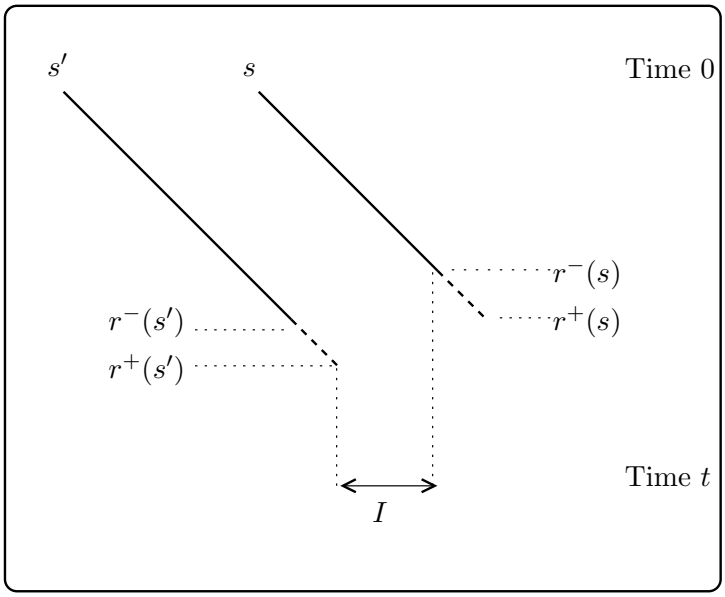


Figure 13: *An illustration for the check performed in Step 5. The solid segments indicate that the object is moving, and the dashed segments indicate that the object is either moving or already stopped. The interval I must fully consist of standing objects at time t.*

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 $\mathrm{poly}(\epsilon)\cdot n$ 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 $\mathrm{poly}(\epsilon)$-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 $\mathtt{ENV}_0(i), \mathtt{ENV}_1(i+\delta), ..., \mathtt{ENV}_n(i+\delta n)$) 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 $\mathtt{ENV}_0(i), \mathtt{ENV}_1(i)..., \mathtt{ENV}_n(i)$ 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 $\mathtt{ENV}'$ that is ϵ_1-close to $\mathtt{ENV}$, for $\epsilon_1 = \mathrm{poly}(\epsilon)$ to be determined later. Indeed, in $\mathtt{ENV}'$ the initial configuration (i.e., $\mathtt{ENV}'_0$) contains no adjacent moving objects, and the movement of individual objects in $\mathtt{ENV}'$ is continuous; that is, $\mathtt{ENV}'$ 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 $\mathtt{ENV}'$ 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 $\mathtt{ENV}'$ (rather than the approximate locations viewed in the sample of $\mathtt{ENV}$) 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 $\texttt{ENV}$. 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 $\texttt{ENV}$). 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 6.1 (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*

$$\Delta(P,Q) = \min_{\pi \in \mathrm{Sym}(n)} \left\{ \sum_{i \in [n]} \|p_i - q_{\pi(i)}\| \right\} \tag{12}$$

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 6.2 (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 6.3).

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 $\mathrm{poly}(m)$-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 6.3 (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. (12) 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*

[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''.

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 6.2. 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 $\text{poly}(\epsilon) \cdot n$. (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 $\text{poly}(\epsilon) \cdot n$ 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 $\text{poly}(\epsilon) \cdot n$. (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 $\text{poly}(\epsilon) \cdot n$, 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, $\texttt{ENV}''$ retains the foregoing features of $\texttt{ENV}'$, and in addition the stopping places of its diagonal lines match the starting places of vertical lines. In other words, $\texttt{ENV}''$ 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.)

Inferring from the success of Step 4. Considering the environment $\texttt{ENV}''$, 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

[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.

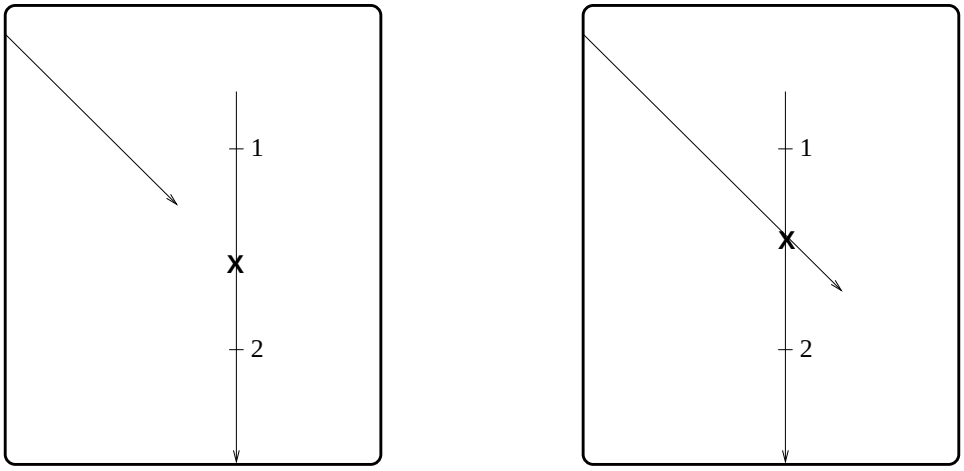


Figure 14: *Analysis of Step 3. The left image shows (part of) a diagonal line that stops too early (with respect to its matched vertical line, which starts at either location 1 or location 2), whereas the right image shows a diagonal that stops too late.*

$s_1 < s_2 < \cdots < s_{n'}$, 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 $\texttt{ENV}'''$. 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 $t'_i - t_{\pi(i)} = s_i - s_{\pi(i)}$.

- 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

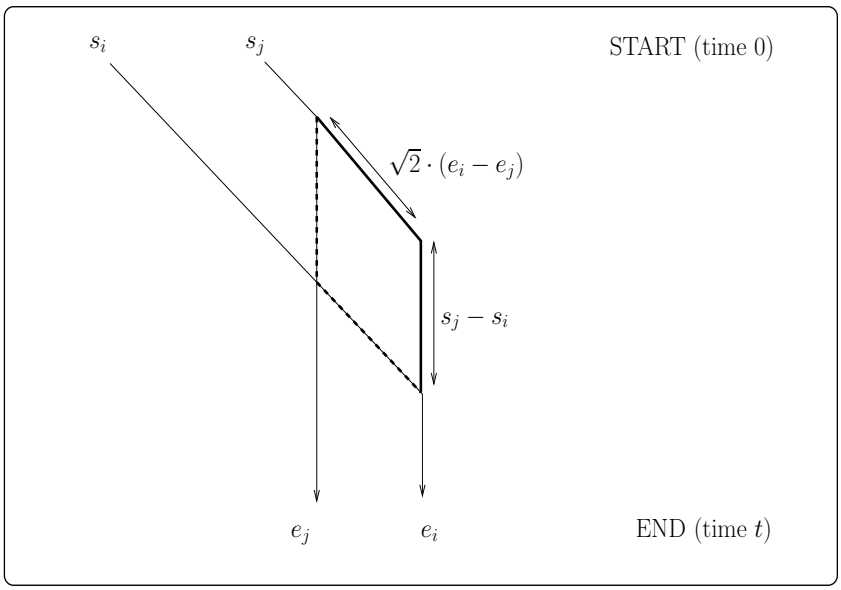


Figure 15: *Eliminating a crossing between lines i and j in the analysis of Step 4 for the case that $\pi(i) = j$ and $\pi(j) = i$. The segments that are omitted are shown in dashes, and the segments that are added are shown in bold.*

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 6.4 (distances versus ranking with respect to sorted order on the line): *Let $r_1 < r_2 < \cdots < r_n$ 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 6.4 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 6.4 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 $j \neq k$

[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 $\texttt{ENV}$ from $\texttt{ENV}''$, which is ϵ_2-close to it.

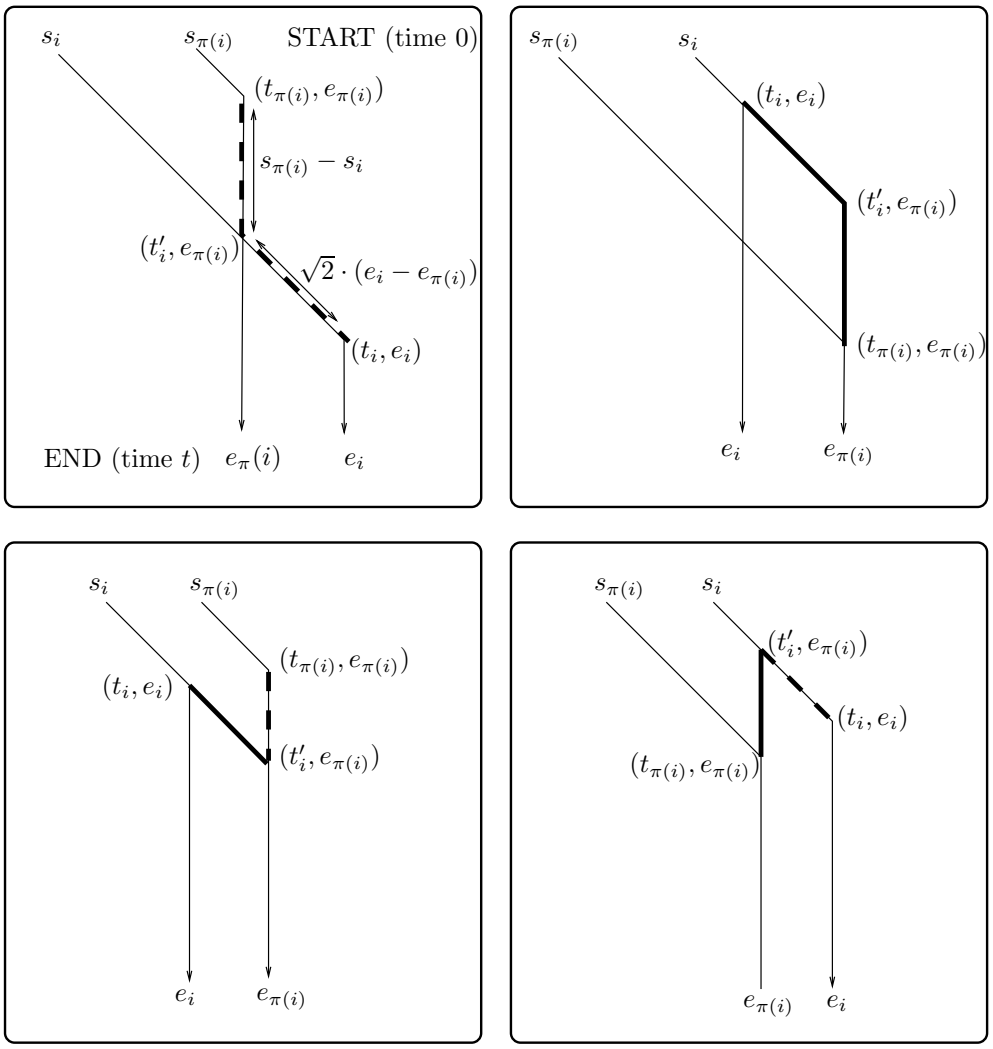


Figure 16: *An illustrations for the charging rule in the analysis of Step 4. The omitted segments are shown in dashes and the added segments are shown bold. In all cases line i is charged for either the omission or the addition of the diagonal segment whose endpoints are (t_i, e_i) and $(t'_i, e_{\pi(i)})$ 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.*

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,

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

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

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 $\mathtt{ENV}'''$ denote the resulting environment and note that $\mathtt{ENV}'''$ is $2\epsilon_3$-close to $\mathtt{ENV}$. 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 6.5 (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 6.5 asserts that if this number must be big (i.e., bigger than $2\epsilon'' n$), then there are many (i.e., $\mathrm{poly}(\epsilon'\epsilon'') \cdot n^2$) 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 6.5, 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 6.5 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 6.5, 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.

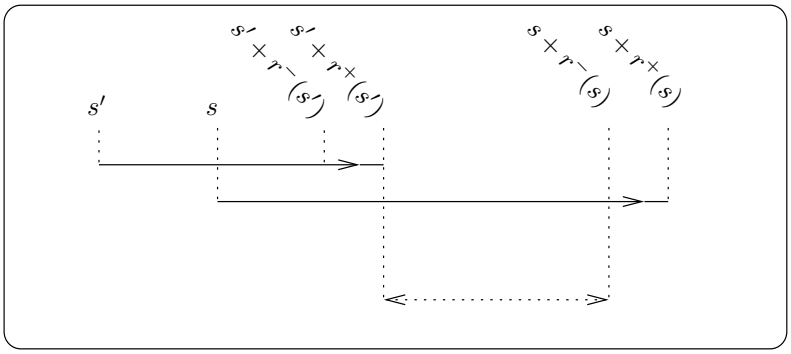


Figure 17: *The intervals considered in Assumption 3. The two arrows depict the actual movement of two objects, starting at locations $s' < s$. The actual stopping times are within the intervals $(s'+r^-(s'), s'+r^+(s')]$ and $(s+r^-(s), s+r^+(s)]$, respectively. The interval $[s'+r^+(s'), s+r^-(s)+1]$ 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 $\texttt{ENV}^\dagger_t(p) = \perp$ 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 $\texttt{ENV}^\dagger$. We next show that conditioned on the above assumptions as well as on the assumption that Step 5 passes successfully, we can transform $\texttt{ENV}^\dagger$ 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 $\texttt{ENV}^\dagger$.)

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.

  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

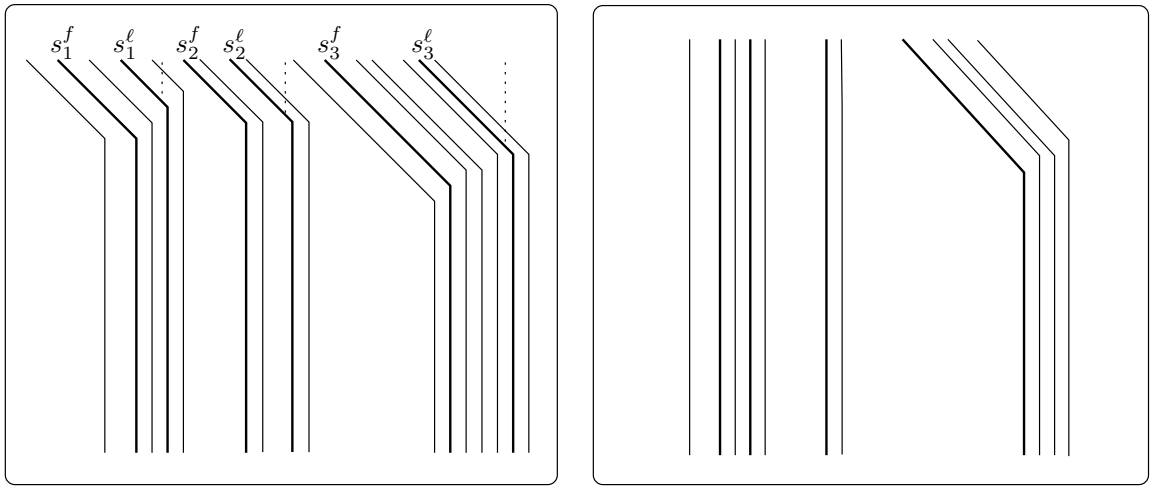


Figure 18: *An illustration for modifications performed in Stage I of the analysis of Step 5. The left image shows (part of) the environment* $\texttt{ENV}^\dagger$ *before lines are removed or "straightened", whereas the right image shows the environment after these changes. The illustration shows three 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.*

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 $\texttt{ENV}_t^\ddagger(p) = \perp$. 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.

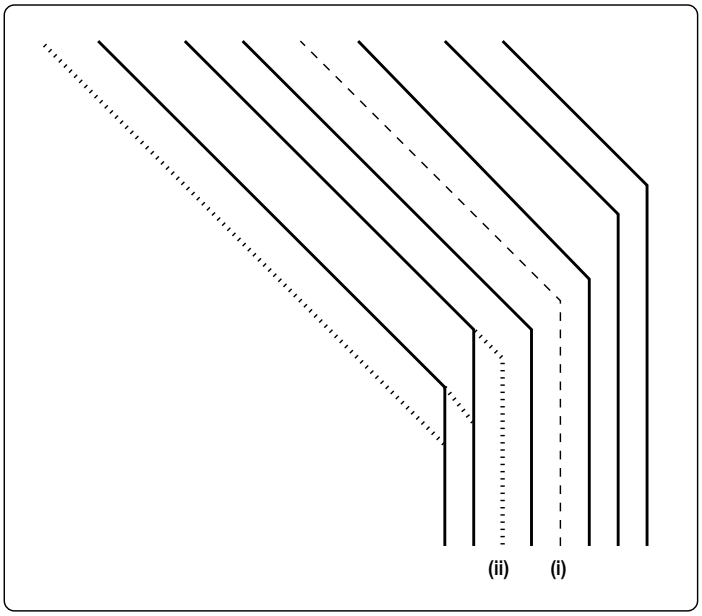


Figure 19: *Filling up the vertical gaps in the analysis of Step 5. The original lines are shown in solid, the dashed line at location (i) is inserted without conflicts, whereas the dotted lines show an insertion at location (ii) and its propagated corrections.*

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 1.8 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).

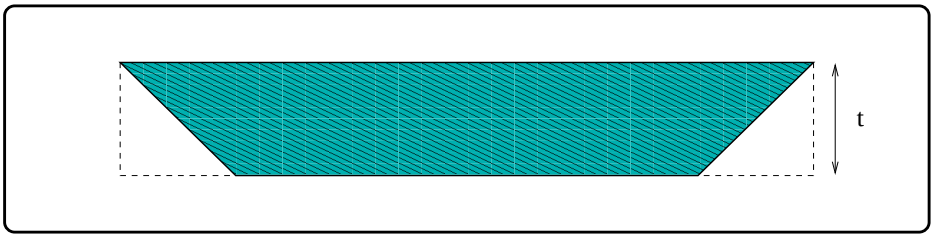


Figure 20: *The trapezoid environments used for the case of $t < \epsilon n/2$.*

Conclusion: Deriving Theorem 1.8 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.

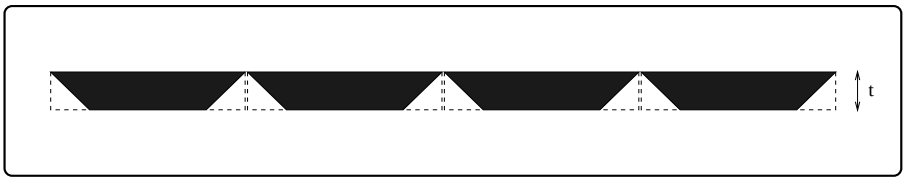


Figure 21: *Tiling a rectangle with trapezoids (for the case of $t < \epsilon n/2$).*

Note that any t-by-n environments that evolves according to the moving-object 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-object 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 1.8 also for this case; that is, we have proved the following.

Theorem 6.6 (Theorem 1.8, restated): *There exists a time-conforming oracle machine of* (total)

time complexity $\mathrm{poly}(1/\epsilon)$ *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 6.6) 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 6.6 (i.e., having complexity that does not depend on n).

Theorem 6.7 (lower bound on one-sided testers): *Any nonadaptive* one-sided error *tester for the consistency of environments of the form* $\mathtt{ENV} : \{0,1,...,n\} \times [n] \to \{-1,0,1,\perp\}$ *with the moving object evolution rule has* (total) *query complexity* $\Omega(\sqrt{n})$.

Before proving Theorem 6.7, we discuss its implications. First note that Theorem 6.7 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 6.6 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 and total time complexity $\mathrm{poly}(\epsilon^{-1}\log n)$. For example, one may replace Step 3 in the foregoing tester by sampling $\mathrm{poly}(\epsilon^{-1})$ 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.

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.

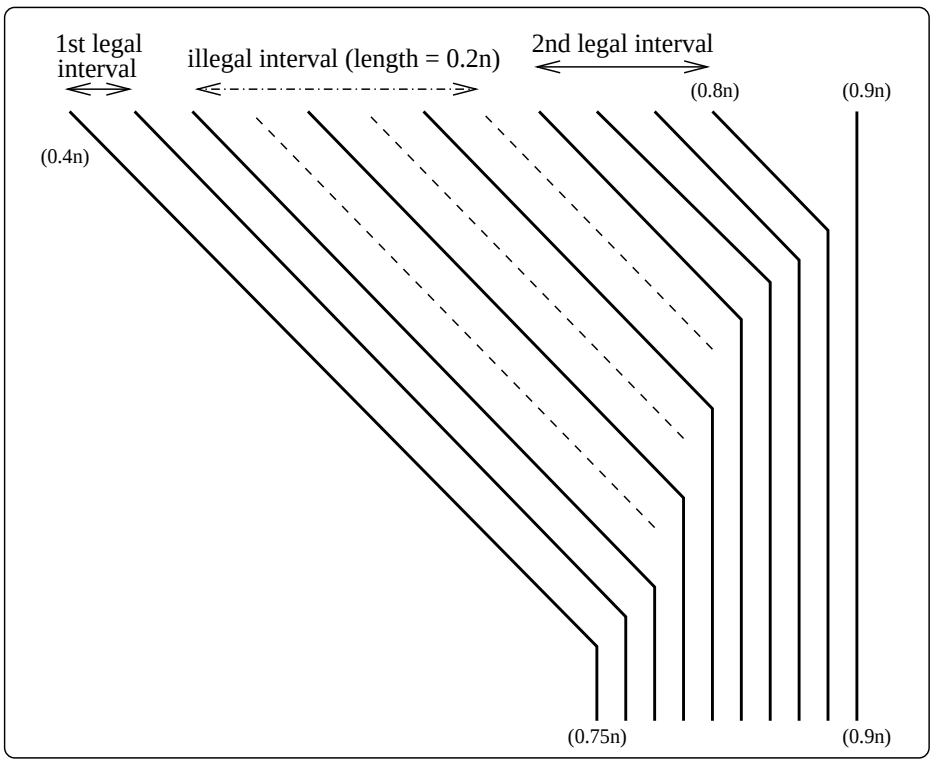


Figure 22: *A generic environment in the family used in the proof of Theorem 6.7. The three dashed lines are illegal (since they have no vertical continuation at the point that they stop). Only a portion of the environment is shown, say* $[0, 0.4n] \times [0.4n, 0.9n]$.

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 $\mathtt{ENV}^{(r)}$ into a legal $\widetilde{\mathtt{ENV}}^{(r)}$. One observation is that we can remove from $\mathtt{ENV}^{(r)}$ any illegal diagonal segment that is not hit by any query. Furthermore, we may can remove from $\mathtt{ENV}^{(r)}$ 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,

$$D(Q) \stackrel{\text{def}}{=} \{s \in [n] : \exists j \in [[n]] \text{ s.t. } (j, s+j) \in Q\}. \tag{13}$$

- 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,

$$G(S) \stackrel{\text{def}}{=} \{0.8n-4i+1, 0.8n-4i+3 : i \in P(S)\} \tag{14}$$

$$\text{where } P(S) \stackrel{\text{def}}{=} \{i \in [n'] : \text{state}_i > 0\}. \tag{15}$$

Fact 6.7.1 *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 6.7.2 *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 6.7.4.

Claim 6.7.3 *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 6.7.1, 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:

$$\begin{aligned}
&\Pr_{r\in[0.1n]}\left[e_r(G(D(Q)\cap I_r)\cup A_{r,|Q|})\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \\
&\quad\leq \Pr_{r\in[0.1n]}\left[e_r(G(D(Q))\cap I_r)\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \qquad (16)\\
&\qquad+\Pr_{r\in[0.1n]}\left[e_r(A_{r,|Q|})\cap(C(Q)\cup\{0.75n\})\neq\emptyset\right] \qquad (17)
\end{aligned}$$

We start with Eq. (17). By the definition of $e_r(\cdot)$, it holds that $e_r(A_{r,|Q|}) = \{0.55n + r - |Q|, ..., 0.55n + r\}$. Hence Eq. (17) 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. (16), 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. (16) 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 6.7.2. The claim follows. ■

Next, we show that if the event referred to in Claim 6.7.3 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 $\texttt{ENV}^{(r)}$ 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 $\texttt{ENV}_Q^{(r)}$ to denote the restriction of the environment $\texttt{ENV}^{(r)}$ to Q. In other words, $\texttt{ENV}_Q^{(r)}$ is determined by the answers to the queries in Q when the environment is $\texttt{ENV}^{(r)}$.

Claim 6.7.4 *Let $A_{r,s}$ be as in Claim 6.7.3. If $e_r(G(D(Q) \cap I_r) \cup A_{r,|Q|}) \cap (C(Q) \cup \{0.75n\}) = \emptyset$, then $\texttt{ENV}_Q^{(r)}$ is consistent with a legal environment of the moving object evolution rule.*

Proof: Given a partial environment $\texttt{ENV}_Q^{(r)}$ 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 $\widetilde{\texttt{ENV}}^{(r)}$ such that $\widetilde{\texttt{ENV}}_Q^{(r)} = \texttt{ENV}_Q^{(r)}$. 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 $s_i = s_{i-1} - 6$.
     (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 $\texttt{ENV}^{(r)}$.

[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.

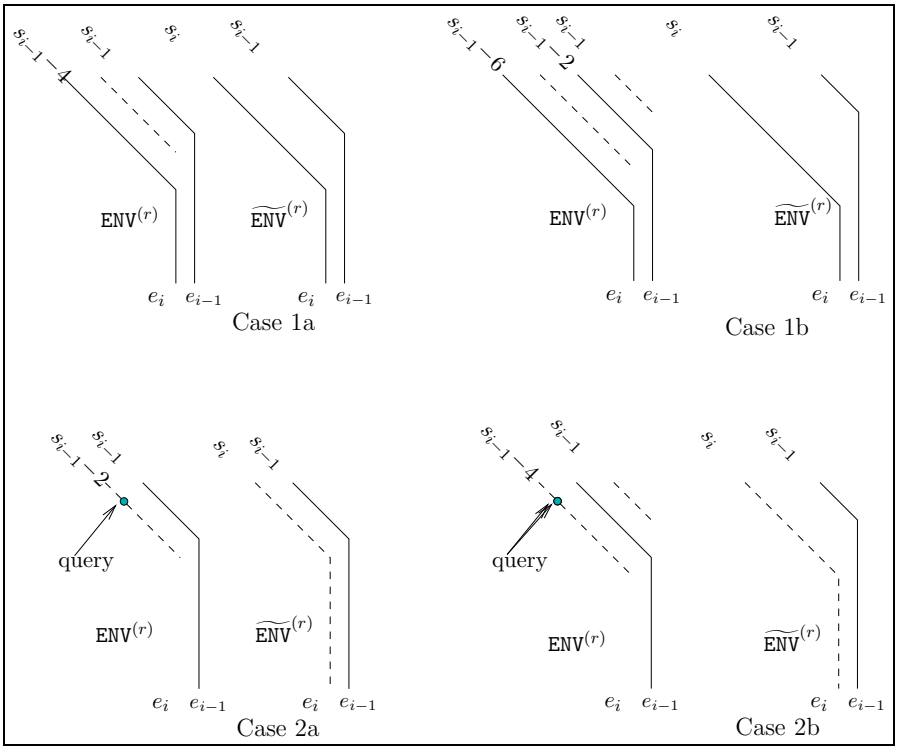


Figure 23: *Detail for the proof of Claim 6.7.4. An illustration for the way the matching is constructed in the illegal region when the queue is empty. As in Figure 22, in the images for* $\mathtt{ENV}^{(r)}$*, the dashed lines represent illegal diagonals (in* $\mathtt{ENV}^{(r)}$*), whereas the solid lines represent legal diagonal and vertical segments.*

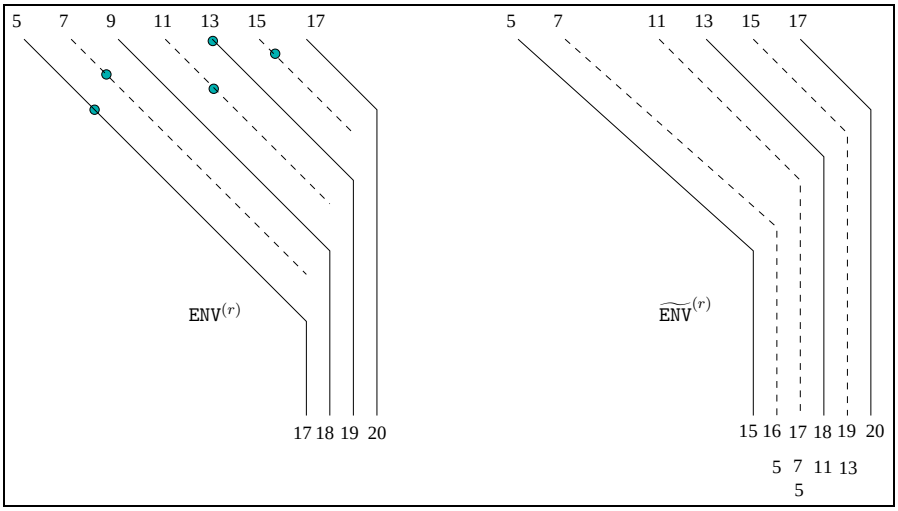
2. The following two cases corresponds to the presence of a query on the relevant diagonal (determined by the case).
   (a) If s_{i-1} corresponds to a legal diagonal and there is a query on the illegal diagonal corresponding to $s_{i-1} - 2$, then we set $s_i = s_{i-1} - 2$. If there is also a query 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 $\mathtt{ENV}^{(r)}$.)

  If there is a diagonal starting in position $(0, s)$ such that $s < s_i$ and in $\mathtt{ENV}^{(r)}$ 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]

[37] Recall that in $\mathtt{ENV}^{(r)}$ 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 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 6.7.4. An illustration for the way the matching is constructed when the queue is non-empty.*

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 $s_i = s_{i-1} - 2$.

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 $\widetilde{\texttt{ENV}}^{(r)}_Q$ and $\texttt{ENV}^{(r)}_Q$ 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 6.7.3 and 6.7.4, 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 $\mathtt{ENV}_Q^{(r)}$ 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 $\mathtt{ENV}^{(r)}$ 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 $\mathtt{ENV}^{(r)}$'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. ■

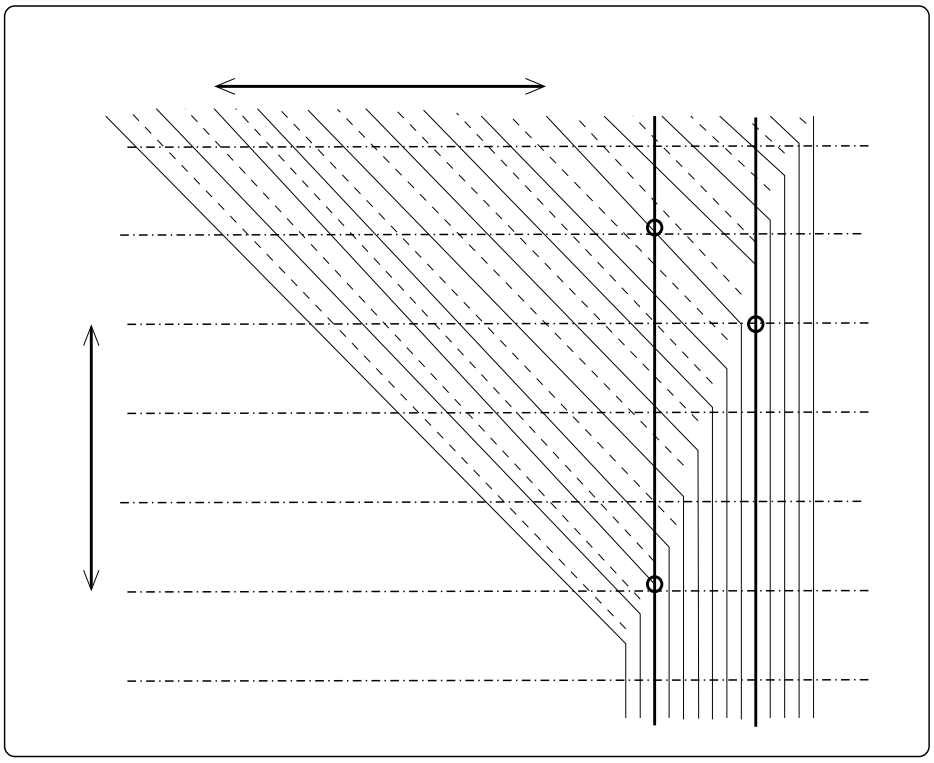


Figure 25: *An illustration for Remark 6.8. The figure shows diagonals that were included in the first set of queries and two vertical lines who's intersections with the horizontal lines were included in the second set of queries. The three small cycles indicate intersection points that (together with 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 6.8 *The construction used in the proof of Theorem 6.7 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 $\texttt{ENV} : [n]^2 \to \{-1, 0, 1, \perp\}$ 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 6.9 *As stated above, we do not know whether the result of Theorem 6.7 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* $\texttt{ENV} : [n]^2 \to \{-1, 0, 1, \perp\}$ *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 6.6 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 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 6.10 *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)*.*

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

[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.

[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.

(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 model of fully visible state, considered in Section 4. That is, for any $d \geq 1$, testing consistency of the evolution of d-dimensional environments with fully visible state 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 state.[40] Details follow.

For any evolution rule $\Gamma : \Sigma^{3^d} \rightarrow \Sigma$, 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 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 stateful objects may not be easier to handle than the general evolution of environments with fully visible state. Still, one may seek natural classes of such variable motion that are easier to handle.

Open Problem 6.11 (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:*

[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.

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 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 $\mathtt{ENV}_j : [n]^d \to \Sigma$). 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 $j-1$ 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 Section 6) 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 $\Sigma^{n^{3^d}}$. Average-case complexity may also be applied to the testing task, but one must be very careful regarding the choice of distributions (cf. [Gol07], 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).[41] 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.

[41] A related collection of extended abstracts and surveys has appeared as [Gol10].

We are grateful to Benny Applebaum for collaboration in early stages of this research.

References

[AIK10] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography by cellular automata or how fast can complexity emerge in nature? In *Proceedings of the First Symposium on Innovations in Computer Science (ICS)*, pages 1–19, 2010.

[BBM12] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[Gol] O. Goldreich. Short locally testable codes and proofs: A survey in two parts. In [Gol10].

[Gol07] O. Goldreich. On the average-case complexity of property testing. Technical Report TR07-057, Electronic colloquium on computational complexity (ECCC), 2007.

[Gol10] O. Goldreich, editor. *Property Testing: Current Research and Surveys*. Springer, 2010. LNCS 6390.

[Gol13] O. Goldreich. On the communication complexity methodology for proving lower bounds on the query complexity of property testing. Technical Report TR13-073, Electronic colloquium on computational complexity (ECCC), 2013.

[GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GR11] O. Goldreich and D. Ron. On proximity oblivious testing. *SIAM Journal on Computing*, 40(2):534–566, 2011.

[GR13] T. Gur and R. Rothblum. Non-interactive proofs of proximity. Technical Report TR13-078, Electronic colloquium on computational complexity (ECCC), 2013.

[GS06] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost linear length. *Journal of the ACM*, 53(4):558–655, 2006.

[KS92] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Math*, 5(4):545–557, 1992.

[KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing (STOC)*, pages 80–86, 2000.

[KV94] M. Kearns and U. Vazirani. *An introduction to Computational Learning Theory*. MIT Press, 1994.

[Ron10] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5:73–205, 2010.

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RS06] S. Raskhodnikova and A. Smith. A note on adaptivity in testing properties of bounded degree graphs. Technical Report TR06-089, Electronic Colloquium on Computational Complexity (ECCC), 2006.

[Val84] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, November 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.[42] These implementation are used for proving Theorem 3.2 (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 till 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 neighbor has done so, and otherwise they forward the token to that neighbor. The entire process takes $O(n')$ steps.

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

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. 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).

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 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 slot.

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 neighboring object that wishes to move to it, then permission is granted to one of these objects (according to a predetermined order of preference). (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 no neighboring object wishes to move to it, then it 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.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de