Hitting Families of Schedules for Asynchronous Programs[⋆]

Dmitry Chistikov[⋆⋆], Rupak Majumdar, and Filip Niksic

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany
{dch,rupak,fniksic}@mpi-sws.org

Abstract. We consider the following basic task in the testing of concurrent systems. The input to the task is a partial order of events, which models actions performed on or by the system and specifies ordering constraints between them. The task is to determine if some scheduling of these events can result in a bug. The number of schedules to be explored can, in general, be exponential.

Empirically, many bugs in concurrent programs have been observed to have small bug depth; that is, these bugs are exposed by every schedule that orders d specific events in a particular way, irrespective of how the other events are ordered, and d is small compared to the total number of events. To find all bugs of depth d, one needs to only test a *d-hitting family* of schedules: we call a set of schedules a d-hitting family if for each set of d events, and for each allowed ordering of these events, there is some schedule in the family that executes these events in this ordering. The size of a d-hitting family may be much smaller than the number of all possible schedules, and a natural question is whether one can find d-hitting families of schedules that have small size.

In general, finding the size of optimal d-hitting families is hard, even for $d = 2$. We show, however, that when the partial order is a tree, one can explicitly construct d-hitting families of schedules of small size. When the tree is balanced, our constructions are polylogarithmic in the number of events.

1 Introduction

Consider the following basic task in systematic testing of programs. We are given n events a_1, a_2, …, a_n, and we ask if the execution of some ordering of these events can cause the program to exhibit a bug. In the worst case, one needs to run $n!$ tests, one corresponding to each ordering of events. Empirically, though, many bugs in programs depend on the precise ordering of a small number of events [13,16,3]. That is, for many bugs, there is some constant d (called the *bug depth*, small in comparison to n) and a subset a_{i_1}, …, a_{i_d} of events such that some ordering of these d events already exposes the bug no matter how

⋆ This research was funded in part by the ERC Synergy award (IMPACT).

⋆⋆ Present address: Department of Computer Science, University of Oxford, UK.

all other events are ordered. This empirical observation is the basis for many different systematic testing approaches such as context-bounded testing [14], delay-bounded testing [6], and PCT [3]. Can we do better than $n!$ tests if we only want to uncover all bugs of depth up to d, for fixed d? An obvious upper bound on the number of tests is given by

$$\binom{n}{d} \cdot d! \leq n^d,$$

which picks a test for each choice of d events and each ordering of these events. In this paper, we show that one can do significantly better—in this as well as in more general settings.

Hitting families of schedules. We consider a more general instance of the problem, where there is a partial ordering between the n events. A *schedule* is a linearization (a linear extension) of the partial order of events. A dependency between two events a and b in the partial order means that in any test, the event a must execute before b. For example, a may be an action to open a file and b an action that reads from the file, or a may be a callback that enables the callback b.

The *depth* of a bug is the minimum number of events that must be ordered in a specific way for the bug to be exposed by a schedule. For example, consider some two events a and b in the partial order of an execution. If a bug manifests itself only when a occurs before b, the bug depth is 2. If there are three events that must occur in a certain order for a bug to appear, the depth is 3, and so on. For example, an order violation involving two operations is precisely a bug of depth 2: say, event a writes, event b reads, or vice versa (race condition). Basic atomicity violation bugs are of depth 3: event a establishes an invariant, b breaks it, c assumes the invariant established by a; bugs of larger depth correspond to more involved scenarios and capture more complex race conditions. A schedule is said to *hit* a bug if the events that expose the bug occur in the schedule in the required order. The question we study in this paper is whether it is possible to find a family of schedules that hits all potential bugs of depth d, for a fixed $d \geq 2$ —we call such a family a *d-hitting family* of schedules.

For a general partial order, finding an optimal d-hitting family is NP-hard, even when $d = 2$ [24]; in fact, even approximating the optimal size is hard [9,4]. Thus, we focus on a special case: when the Hasse diagram of the partial order is a tree. Our choice is motivated by several concurrent programming models, such as asynchronous programs [20,11,8] and JavaScript events [17], whose execution dependencies can be approximated as trees.

Constructing hitting families for trees. For trees and $d = 2$, it turns out that two schedules are enough, independent of the number of events in the tree. These two schedules correspond to leftmost and rightmost DFS (depth-first) traversals of the tree, respectively.

For $d > 2$ and an execution tree of n events, we have already mentioned the upper bound of n^d for the size of an optimal d-hitting family (cf. delay-bounded scheduling [6]). Our main technical results show that this family can be exponentially sub-optimal. For $d = 3$ and a balanced tree on n nodes, we show an explicit construction of a 3-hitting family of size $O(\log n)$, which is optimal up to a constant factor. (Our construction works on a more general partial order, which we call a *double tree.*) For each $d > 3$, we show an explicit construction of a d-hitting family of size $f(d) \cdot (\log n)^{d-1}$, which is optimal up to a polynomial. Here $f(d)$ is an exponential function depending only on d. As a corollary, the two constructions give explicit d-hitting families of size $O(\log n)$ (for $d = 3$) and $O((\log n)^{d-1})$ (for $d > 3$) for antichains, i.e., for the partial order that has no dependencies between the n events. We also show a lower bound on the size of d-hitting families in terms of the height of the tree; in a dual way, for an antichain of n events, the size of any d-hitting family is at least $g(d) \cdot \log n$ for each $d > 2$.

For a testing scenario where the *height* of the tree (the size of the maximum chain of dependencies) is exponentially smaller than its *size* (the number of events), our constructions give explicit test suites that are exponentially smaller than the size—in contrast to previous techniques for systematic testing.

Related work. Our notion of bug depth is similar to bug depth for shared-memory multi-threaded programs introduced in [3]. The quantity in [3] is defined as the minimal number of additional constraints that guarantee an occurrence of the bug. Depending on the bug, this can be between half our d and one less than our d. Burckhardt et al. [3] show an $O(mn^{d'-1})$ family for m threads with n instructions in total (d' denotes bug depth according to their definition). Since multi-threaded programs can generate arbitrary partial orders, it is difficult to prove optimality of hitting families in this case.

Our notion of d-hitting families is closely related to the notion of *order dimension* for a partial order, defined as the smallest number of linearizations, the intersection of which gives rise to the partial order [5,23,19]. Specifically, the size of an optimal 2-hitting family is the order dimension of a partial order, and the size of an optimal d-hitting family is a natural generalization. To the best of our knowledge, general d-hitting families have not been studied before for general partial orders. A version of the dimension ($d = 2$) called fractional dimension is known to be of use for approximation of some problems in scheduling theory [2]. Other generalizations of the dimension are also known (see, e.g., [22]), but, to the best of our knowledge, none of them is equivalent to ours.

Summary. The contribution of this paper is as follows:

- We introduce d-hitting families as a common framework for systematic testing (Section 2). The size of optimal d-hitting families generalizes the order dimension for partial orders, and the families themselves are natural combinatorial objects of independent interest.
- We provide explicit constructions of d-hitting families for trees that are close to optimal: up to a small constant factor for $d = 3$ and up to a polynomial for

$d > 3$ (Sections 3–5). Our families of schedules can be exponentially smaller than the size of the partial order.

We outline some challenges in going from our theoretical constructions to building practical and automated test generation tools in Section 6.

2 Hitting families of schedules

In this section, we first recall the standard terminology of partial orders, and then proceed to define schedules (linearizations of these partial orders) and hitting families of schedules.

Preliminaries: Partial orders. A *partial order* (also known as a partially ordered set, or a poset) is a pair $(\mathcal{P}, \leq)$ where $\mathcal{P}$ is a set and $\leq$ is a binary relation on $\mathcal{P}$ that is:

1) reflexive: $x \leq x$ for all $x \in \mathcal{P}$,
2) antisymmetric: $x \leq y$ and $y \leq x$ imply $x = y$ for all $x, y \in \mathcal{P}$,
3) transitive: $x \leq y$ and $y \leq z$ imply $x \leq z$ for all $x, y, z \in \mathcal{P}$.

One typically uses $\mathcal{P}$ to refer to $(\mathcal{P}, \leq)$. We will refer to elements of partial orders as *events*; the *size* of $\mathcal{P}$ is the number of events in it, $|\mathcal{P}|$.

The relation $x \leq y$ is also written as $x \leq_{\mathcal{P}} y$ and as $y \geq x$; the event x is a *predecessor* of y, and y is a *successor* of x. One writes $x < y$ iff $x \leq y$ and $x \neq y$. Furthermore, x is an *immediate predecessor* of y (and y is an *immediate successor* of x) if $x < y$ but there is no $z \in \mathcal{P}$ such that $x < z < y$. The *Hasse diagram* of a partial order $\mathcal{P}$ is a directed graph where the set of vertices is $\mathcal{P}$ and an edge (x, y) exists if and only if x is an immediate predecessor of y. Partial orders are sometimes identified with their Hasse diagrams.

Events x and y are *comparable* iff $x \leq y$ or $y \leq x$. Otherwise they are *incomparable*, which is written as $x \shortparallel y$. Partial orders $(\mathcal{P}_1, \leq_1)$ and $(\mathcal{P}_2, \leq_2)$ are *disjoint* if $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$; the *parallel composition* (or *disjoint union*) of such partial orders is the partial order $(\mathcal{P}, \leq)$ where $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ and $x \leq y$ iff $x, y \in \mathcal{P}_k$ for some $k \in \{1, 2\}$ and $x \leq_k y$. In this partial order, which we will denote by $\mathcal{P}_1 \parallel \mathcal{P}_2$, any two events not coming from a single $\mathcal{P}_k$ are incomparable: $x_1 \in \mathcal{P}_1$ and $x_2 \in \mathcal{P}_2$ imply $x_1 \shortparallel x_2$.

For a partial order $(\mathcal{P}, \leq)$ and a subset $\mathcal{Q} \subseteq \mathcal{P}$, the *restriction* of $(\mathcal{P}, \leq)$ to $\mathcal{Q}$ is the partial order $(\mathcal{Q}, \leq_{\mathcal{Q}})$ in which, for all $x, y \in \mathcal{Q}$, $x \leq_{\mathcal{Q}} y$ if and only if $x \leq y$. Instead of $\leq_{\mathcal{Q}}$ one usually writes $\leq$, thus denoting the restriction by $(\mathcal{Q}, \leq)$. We will also say that the partial order $\mathcal{P}$ *contains* the partial order $\mathcal{Q}$. In general, partial orders $(\mathcal{P}_1, \leq_1)$ and $(\mathcal{P}_2, \leq_2)$ are *isomorphic* iff there exists an isomorphism $f \colon \mathcal{P}_1 \to \mathcal{P}_2$: a bijective mapping that respects the ordering, i.e., with $x \leq_1 y$ iff $f(x) \leq_2 f(y)$ for all $x, y \in \mathcal{P}_1$. Containment of partial orders is usually understood up to isomorphism.

Schedules and their families. A partial order is *linear* (or total) if all its events are pairwise comparable. A linearization (linear extension) of the partial order $(\mathcal{P}, \leq)$ is a partial order of the form $(\mathcal{P}, \leq')$ that is linear and has $\leq'$ which is a superset of $\leq$. We call linearizations (linear extensions) of $\mathcal{P}$ *schedules*. In other words, a schedule α is a permutation of the elements of $\mathcal{P}$ that *respects* $\mathcal{P}$, i.e., *respects* all constraints of the form $x \leq y$ from $\mathcal{P}$: for all pairs $x, y \in \mathcal{P}$, whenever $x \leq_{\mathcal{P}} y$, it also holds that $x \leq_\alpha y$. We denote the set of all possible schedules by $S(\mathcal{P})$; a *family* of schedules for $\mathcal{P}$ is simply a subset of $S(\mathcal{P})$.

In what follows, we often treat schedules as words and families of schedules as languages. Indeed, let $\mathcal{P}$ have n elements $\{v_1, \ldots, v_n\}$, then any schedule α can be viewed as a word of length n over the alphabet $\{v_1, \ldots, v_n\}$ where each letter occurs exactly once. We say that α *schedules* events in the order of occurrences of letters in the word that represents it.

Suppose α_1 and α_2 are schedules for disjoint partial orders $\mathcal{P}_1$ and $\mathcal{P}_2$; then $\alpha_1 \cdot \alpha_2$ is a schedule for the partial order $\mathcal{P}_1 \parallel \mathcal{P}_2$ that first schedules all events from $\mathcal{P}_1$ according to α_1 and then all events from $\mathcal{P}_2$ according to α_2. Note that we will use the $\cdot$ to concatenate schedules (as well as individual events); since some of our partially ordered sets will contain strings, concatenation "inside" an event will be denoted simply by juxtaposition.

Admissible tuples and *d*-hitting families. Fix a partial order $\mathcal{P}$ and let $\boldsymbol{a} = (a_1, \ldots, a_d)$ be a tuple of $d \geq 2$ distinct elements of $\mathcal{P}$; we call such tuples *d-tuples*. Suppose α is a schedule for $\mathcal{P}$; then the schedule α *hits* the tuple $\boldsymbol{a}$ if the restriction of α to the set $\{a_1, \ldots, a_d\}$ is the sequence $a_1 \cdot \ldots \cdot a_d$.

Note that for a tuple $\boldsymbol{a}$ to have a schedule that hits $\boldsymbol{a}$ it is necessary and sufficient that $\boldsymbol{a}$ respect $\mathcal{P}$; this condition is equivalent to the condition that $a_i \leq a_j$ or $a_i \parallel a_j$ whenever $1 \leq i \leq j \leq d$. We call d-tuples satisfying this condition *admissible.*

Definition 1 (*d*-hitting family). A family of schedules F for $\mathcal{P}$ is *d-hitting* if for every admissible d-tuple $\boldsymbol{a}$ there is a schedule $\alpha \in F$ that hits $\boldsymbol{a}$.

It is straightforward that every $\mathcal{P}$ with $|\mathcal{P}| = n$ has a d-hitting family of size at most $\binom{n}{d} \cdot d! \leq n^d$: just take any hitting schedule for each admissible d-tuple, of which there are at most $\binom{n}{d} \cdot d!$. For $d = 2$, the size of the smallest 2-hitting family is known as the dimension of the partial order [5,23]. Computing and even approximating the dimension for general partial orders is known to be a hard problem [24,9,4]. In the remainder of the paper, we focus on d-hitting families for specific partial orders, most importantly trees (which can, for instance, approximate happens-before relations of asynchronous programs). We first consider two simple examples.

Example 2 (chain). Consider a chain of n events (a linear order): $\mathcal{C}_n = \{1, \ldots, n\}$ with $1 < 2 < \ldots < n$. This partial order has a unique schedule: $\alpha = 1 \cdot 2 \cdot \ldots \cdot n$; a d-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ is admissible iff $a_1 < \ldots < a_d$, and α hits all such d-tuples. Thus, for any d, the family $F = \{\alpha\}$ is a d-hitting family for $\mathcal{C}_n$.

Example 3 (chain with independent event). Consider $\mathcal{C}_n \parallel \{\dagger\}$, the disjoint union of $\mathcal{C}_n$ from Example 2 and a singleton $\{\dagger\}$. There are $n+1$ possible schedules, depending on how $\dagger$ is positioned with respect to the chain: $\alpha_0 = \dagger \cdot 1 \cdot 2 \cdot \ldots \cdot n$, $\alpha_1 = 1 \cdot \dagger \cdot 2 \cdot \ldots \cdot n$, $\ldots$, $\alpha_n = 1 \cdot 2 \cdot \ldots \cdot n \cdot \dagger$. For $d = 2$, admissible pairs are of the form (i, j) with $i < j$, $(\dagger, i)$, and $(i, \dagger)$ for all $1 \le i \le n$; the family $F_2 = \{\alpha_0, \alpha_n\}$ is the smallest 2-hitting family. Now consider $d = 3$. Note that all triples $(i, \dagger, i+1)$ with $1 \le i \le n-1$, as well as $(\dagger, 1, 2)$ and $(n-1, n, \dagger)$, are admissible, and each of them is hit by a unique schedule. Therefore, the smallest 3-hitting family of schedules consists of all $n+1$ schedules: $F_3 = \{\alpha_0, \ldots, \alpha_n\}$. For $d \ge 4$, it remains to observe that every d-hitting family is necessarily d'-hitting for $2 \le d' \le d$, hence F_3 is optimal for all $d \ge 3$.

An important **corollary** of this example is that, for any $d \ge 3$ and any partial order $\mathcal{P}$, every d-hitting family must contain at least $m+1$ schedules, where m denotes the maximum number n such that $\mathcal{P}$ contains $\mathcal{C}_n \parallel \{\dagger\}$. This m is upper-bounded (and this upper bound is tight) by the *height* of the partial order $\mathcal{P}$, sometimes called *length*: the maximal cardinality of a chain (a set of pairwise comparable events) in $\mathcal{P}$.

3 Hitting families of schedules for trees

3.1 Definitions and overview

Consider a complete binary tree of height h with edges directed from the root. This tree is the Hasse diagram of a partial order $\mathcal{T}^h$, unique up to isomorphism; we will apply tree terminology to $\mathcal{T}^h$ itself. The root of $\mathcal{T}^h$ forms the 0th *layer*, its children the 1st layer and so on. The maximum k such that $\mathcal{T}^h$ has an element in the kth layer is the height of the tree $\mathcal{T}^h$. We will assume that elements of $\mathcal{T}^h$ are strings: $\mathcal{T}^h = \{0,1\}^{\le h}$ with $x \le y$ for $x, y \in \mathcal{T}^h$ iff x is a prefix of y. The kth layer of $\mathcal{T}^h$ is $\{0,1\}^k$, and nodes of the hth layer are *leaves.* Unless $x \in \mathcal{T}^h$ is a leaf, nodes $x\,0$ and $x\,1$ are left- and right-children of x, respectively. (Recall that the juxtaposition here denotes concatenation of strings, with the purpose of distinguishing individual strings and their sequences.) The tree $\mathcal{T}^h$ has $n = 2^{h+1} - 1$ nodes.

The central question that we study in this paper is as follows: How big are optimal d-hitting families of schedules for $\mathcal{T}^h$ with n nodes?

As it turns out, for $\mathcal{T}^h$ very efficient constructions of d-hitting families exist. It is, in fact, possible, to find such families that have size *exponentially smaller* than n, the number of events. More specifically, we prove the following results (h is the height of the partial order—the size of the longest chain):

1. For arbitrary $d \ge 3$, there is a simple d-hitting family of size $O(n^{d-2})$ (Claim 5 in the following subsection 3.2).
2. For $d = 3$, there is a 3-hitting family of size $O(h)$ (Theorem 7 in Section 4).
3. For arbitrary $d \ge 3$, there is a d-hitting family of size $O(h^{d-1})$ (Theorem 10 in Section 5).

Our main technical results are Theorems 7 and 10, shown in the next sections—where they are stated for complete binary trees, with $h = \log(n + 1) - 1$. (Arbitrary trees are, of course, contained in these complete trees, and our constructions extend in a natural way.) The remainder of this section is structured as follows. In subsection 3.2, we prove, as a warm-up, Claim 5. After this, in subsection 3.3, we show that the problem of finding families of schedules with size smaller than n turns out to be tricky even when there are no dependencies between events at all. This problem arises as a sub-problem when considering trees (as, indeed, there are no dependencies between the leaves in a tree), and thus our main constructions in Sections 4 and 5 must be at least as agile.

3.2 Warm-up: d-hitting families of size $O(n^{d-2})$

Claim 4. The smallest 2-hitting family of schedules for $\mathcal{T}^h$ has size 2.

The construction is as follows. Take $F_{\mathsf{dfs}} = \{\lambda, \rho\}$ where λ and ρ are left-to-right and right-to-left DFS (depth-first) traversals of $\mathcal{T}^h$, respectively. More formally, these schedules are defined as follows: for $x, y \in \mathcal{T}^h$, $x \leq_\lambda y$ if either $x \leq y$ (i.e., x is a prefix of y) or $x = u\,0\,x'$ and $y = u\,1\,y'$ for some strings $u, x', y' \in \{0, 1\}^*$; $x \leq_\rho y$ if either $x \leq y$ or $x = u\,1\,x'$ and $y = u\,0\,y'$. For instance, $\mathcal{T}^2$ has $\lambda = \varepsilon \cdot 0 \cdot 00 \cdot 01 \cdot 1 \cdot 10 \cdot 11$ and $\rho = \varepsilon \cdot 1 \cdot 11 \cdot 10 \cdot 0 \cdot 01 \cdot 00$. The family F_{dfs} is 2-hitting: all admissible pairs (x, y) satisfy either $x \leq y$, in which case they are hit by any possible schedule, or $x \parallel y$, in which case neither is a prefix of the other, $x = u\,a\,x'$ and $y = u\,\bar{a}\,y'$ with $\{a, \bar{a}\} = \{0, 1\}$, so λ and ρ schedule them in reverse orders. Since it is clear that a family of size 1 cannot be 2-hitting for $\mathcal{T}^h$ with $h \geq 1$ (as $\mathcal{T}^h$ contains at least one pair of incomparable elements), the family F_{dfs} is optimal.

Based on this construction for $d = 2$, it is possible to find d-hitting families for $d \geq 3$ that have size $o(n^d)$ where $n = 2^{h+1} - 1$ is the number of events in $\mathcal{T}^h$:

Claim 5. For any $d \geq 3$, $\mathcal{T}^h$ has a d-hitting family of schedules of size $O(n^{d-2})$.

Indeed, group all admissible d-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$ into bags agreeing on $a_1, \ldots, a_{d-2}$. For each bag, construct a pair of schedules $\lambda' = \lambda'(a_1, \ldots, a_{d-2})$ and $\rho' = \rho'(a_1, \ldots, a_{d-2})$ as follows. In both λ' and ρ', first *schedule* $a_1, \ldots, a_{d-2}$: that is, start with an empty sequence of events, iterate over $k = 1, \ldots, d-2$, and, for each k, append to the sequence all events $x \in \mathcal{T}^h$ such that $x \leq a_k$. The order in which these $x$es are appended is chosen in the unique way that respects the partial order $\mathcal{T}^h$. Events that are predecessors of several a_k are only scheduled once, for the least k. Note that no a_k, $1 \leq k \leq d$, is a predecessor of any a_j for $j < k$, because otherwise the d-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ is not admissible. After this, the events of $\mathcal{T}^h$ that have not been scheduled yet form a disjoint union of several binary trees. The schedule λ' then schedules all events according to how the left-to-right DFS traversal λ would work on $\mathcal{T}^h$, omitting all events that have already been scheduled, and the schedule ρ' does the same based on ρ. As a result, these two schedules hit all admissible d-tuples that agree on $a_1, \ldots, a_{d-2}$; collecting all such schedules for all possible $a_1, \ldots, a_{d-2}$ makes a d-hitting family for $\mathcal{T}^h$ of size at most $2n^{d-2}$.

3.3 Antichains: d-hitting families of size $f(d) \log n$

An *antichain* is a partial order where every two elements are incomparable: $\mathcal{A}_n = \{v_1\} \parallel \{v_2\} \parallel \ldots \parallel \{v_n\}$. The set of all schedules for $\mathcal{A}_n$ is S_n, the set of all permutations, and the set of all admissible d-tuples is the set of all d-arrangements of these n events.

For our problem of finding hitting families of schedules for trees, considering antichains is, in fact, an important subproblem. For example, a complete binary tree with m nodes contains an antichain of size $\lceil m/2 \rceil$: the set of its leaves. Thus, any d-hitting family of sublinear size for the tree must necessarily extend a d-hitting family of sublinear size for the antichain—a problem of independent interest that we study in this section.

Theorem 6. *For any $d \geq 3$, the smallest d-hitting family for $\mathcal{A}_n$ has size between $g(d)\log n - O(1)$ and $f(d)\log n$, where $g(d) \geq d/2\log(d+1)$ and $f(d) \leq d!\, d$.*

We sketch the proof of Theorem 6 in the remainder of this section. We will show how to obtain the upper bound by two different means: with the probabilistic method and with a greedy approach. From the results of the following section 4 one can extract a derandomization for $d = 3$, also with size $O(\log n)$; and section 5 achieves size $f(d) \cdot (\log n)^{d-1}$ for $d \geq 3$. In the current section we also show a lower bound based on a counting argument; the reasoning above demonstrates that this lower bound for antichains extends to a lower bound for trees (see Corollary 8).

Upper bound: Probabilistic method. Consider a family of schedules $F = \{\alpha_1, \ldots, \alpha_k\}$ where each α_i is chosen independently and uniformly at random from S_n; the parameter k will be chosen later. Fix any admissible $\boldsymbol{a} = (a_1, \ldots, a_d)$. What is the probability that a specific α_i does not hit $\boldsymbol{a}$? A random permutation arranges $a_1, \ldots, a_d$ in one of $d!$ possible orders without preference to any of them, so this probability is $1 - 1/d!$. Since all α_i are chosen independently, the probability that none of them hits $\boldsymbol{a}$ is $(1 - 1/d!)^k$. By the union bound, the probability that *at least one* d-tuple $\boldsymbol{a}$ is not hit by any of α_i does not exceed $p = n^d \cdot (1 - 1/d!)^k$.

Now observe that this value of p is exactly the probability that F is not a d-hitting family. If we now choose k in such a way that $p < 1$, then the probability of F being a d-hitting family is non-zero, i.e., a d-hitting family of size k exists. Calculation shows that $k > (d!\, d)\log n / \log e$ suffices.

The probabilistic method, a classic tool in combinatorics, is due to Erdős [1].

Upper bound: Greedy approach. We exploit the following connection between d-hitting families and *set covers*. Recall that in a set cover problem one is given a number of sets, $R_1, \ldots, R_s$, and the goal is to find a small number of these sets whose union is equal to $R = R_1 \cup \ldots \cup R_s$. A set R_i covers an element $e \in R$ iff $e \in R_i$, and this covering is essentially the same as hitting in

d-hitting families: elements $e \in R$ are admissible d-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$, and each schedule α corresponds to a set R_α that contains all d-tuples $\boldsymbol{a}$ that it hits. A d-hitting family of schedules is then the same as a set cover.

A well-known approach to the set cover problem is the greedy algorithm, which in our setting works as follows. Initialize a list of all admissible $\boldsymbol{a} = (a_1, \ldots, a_d)$; on each step, pick some schedule α that hits the largest number of tuples in the list, and cross out all these tuples. Terminate when the list is empty; the set of all picked schedules is a d-hitting family.

While this algorithm can be used for any partial order $\mathcal{P}$, in our case we can estimate the quality of its output. The so-called greedy covering lemma by Sapozhenko [18] or a more widely known Lovász-Stein theorem [12,21] gives an explicit upper bound on the size of the obtained greedy cover in terms of $|R|$ and the density of the instance (the smallest γ such that every $e \in R$ belongs to at least γs out of s sets). In our case, $|R| \le n^d$, and the density is $1/d!$; the obtained upper bound on the size of the smallest d-hitting family is $d!\, d \cdot \log n / \log e - \Theta(d!\, d \log d)$.

Lower bound. Consider the case $d = 3$. Take any 3-hitting family $F = \{\alpha_1, \ldots, \alpha_k\}$ and consider the binary matrix $B = (b_{ij})$ of size $k \times (n-1)$ where $b_{ij} = 1$ iff the schedule α_i places event v_j before v_n. We claim that all columns of B are pairwise distinct. Indeed, if for some $j' \ne j''$ and all i it holds that $b_{ij'} = b_{ij''}$, then no schedule from F can place $v_{j'}$ before v_n without also placing $v_{j''}$ before v_n, and vice versa. This means that no schedule from F hits the 3-tuples $\boldsymbol{a}' = (v_{j'}, v_n, v_{j''})$ and $\boldsymbol{a}'' = (v_{j''}, v_n, v_{j'})$, so F cannot be 3-hitting.

Since all columns of B are pairwise distinct and B is a 0/1-matrix, it follows that the number of columns, $n-1$, cannot be greater than the number of all subsets of its rows, 2^k. From $n - 1 \le 2^k$ we deduce that $k \ge \log(n-1)$. The construction in the general case $d \ge 3$ is analogous.

As we briefly explained above, the lower bound for an antichain of size n remains valid for any partial order that *contains* an antichain of size n (as defined in Section 2). We invoke this argument in Theorem 7 and Corollary 8 in the following section.

4 3-hitting families of size $O(\log n)$

The goal of this section is to construct 3-hitting families of schedules for trees. In fact, the construction that we develop is naturally stated for slightly more involved partial orders, which we call double trees. These double trees are extensions of trees (see Fig. 1). We construct explicit 3-hitting families of schedules of logarithmic size for double trees, so that restriction of these 3-hitting families to appropriate subsets of events gives explicit 3-hitting families for trees and for antichains, also of logarithmic size.

The *(binary) double tree* of half-height $h \ge 1$ is the partial order $\mathcal{D}$ defined as follows. Intuitively, each $\mathcal{D}^h$ is a parallel composition (disjoint union) of two copies of $\mathcal{D}^{h-1}$, with additional top and bottom (largest and smallest) events;

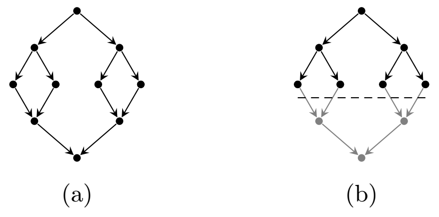

Fig. 1. (a) A double tree ($h = 2$); (b) A tree embedded into a double tree

and the induction basis is that $\mathcal{D}^0$ consists of a single event. Fig. 1 depicts $\mathcal{D}^2$, the double tree of half-height 2.

More precisely, (the Hasse diagram of) $\mathcal{D}$ consists of two complete binary trees of height h that share their set of 2^h leaves; in the first tree, the edges are directed from the root to the leaves, and in the second tree, from the leaves to the root. Formally, $\mathcal{D}^h = \{-1, +1\} \times \{0, 1\}^{\leq h-1} \cup \{0\} \times \{0, 1\}^h$; note that the cardinality of this set is $3 \cdot 2^h - 2$. Each event $x = (s_x, x') \in \mathcal{D}^h$ either belongs to one of the trees ($s_x \in \{-1, +1\}$) or is a shared leaf ($s_x = 0$). We define the ordering by taking the transitive closure of the following relation: let $x = (s_x, x')$ and $y = (s_y, y')$ be events of $\mathcal{D}^h$; if $\{s_x, s_y\} \subseteq \{-1, 0\}$, then $x \leq y$ whenever x' is a prefix of y'; and if $\{s_x, s_y\} \subseteq \{0, +1\}$, then $x \leq y$ whenever y' is a prefix of x'. (Note that all events x, y with $s_x = s_y = 0$ are pairwise incomparable.)

Theorem 7. *The smallest 3-hitting family for the double tree $\mathcal{D}^h$ with $n = 3 \cdot 2^h - 2$ events has size between $2h = 2 \log n - O(1)$ and $4h = 4 \log n - O(1)$.*

Recall that a double tree with $3{\cdot}2^h{-}2$ events contains a complete binary tree with $2{\cdot}2^h{-}1$ nodes, which in turn contains an antichain of size 2^h. As a corollary, $\mathcal{T}^h$, a tree with $n = 2{\cdot}2^h{-}1$ nodes, has a 3-hitting family of size $4h = 4\log(n{+}1){-}4$. Similarly, $\mathcal{A}_n$, an antichain of size $n = 2^h$, has a 3-hitting family of size $4 \log n$. Unlike the constructions from subsection 3.3, the construction of Theorem 7 is explicit.

Corollary 8. *For an arbitrary (not necessarily balanced) tree of height h, out-degree at most Δ, and with at least 2 children of the root, the smallest 3-hitting family has size between h and $4h \log \Delta$.*

Note that lower bounds proportional to h follow from Example 3. We describe the construction of Theorem 7 below.

Matrix notation. We use the following notation for families of schedules. Let $\mathcal{P}$ be a partial order, $|\mathcal{P}| = n$. Let F be a family of schedules for $\mathcal{P}$, $|F| = m$. We then write

$$F = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

where $F = \{\alpha_1, \ldots, \alpha_m\}$ and $\alpha_i = a_{i1} \cdot a_{i2} \cdot \ldots \cdot a_{in}$ for $1 \leq i \leq m$. In other words, a family of m schedules for an n-sized partial order is written as an $m \times n$-matrix whose entries are elements of $\mathcal{P}$, with no element appearing more than once in any row. In particular, if α is a schedule for $\mathcal{P}$, then we represent it with a row vector. The union of families naturally corresponds to stacking of matrices: $F_1 \cup F_2 = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$, and putting two matrices of the same height m next to each other corresponds to concatenating two families of size m, in order to obtain a family of size m for the union of two partial orders: $\begin{pmatrix} F_1 & F_2 \end{pmatrix}$.

Construction of 3-hitting families for double trees. We define the families of schedules using induction on h; in matrix notation, the families will be denoted and structured as follows:

$$M_h = \begin{bmatrix} A_h & B_h \\ C_h & D_h \end{bmatrix}$$

where all four blocks are of size $(3 \cdot 2^{h-1} - 1) \times 2h$; in total, M^h will contain $4h$ schedules, each with $3 \cdot 2^h - 2$ events.

Base case, $h = 1$:

$$\begin{bmatrix} A_1 | B_1 \end{bmatrix} = \begin{bmatrix} C_1 | D_1 \end{bmatrix} = \left[\begin{array}{cc|cc} (-1,\varepsilon) & (0,0) & (0,1) & (+1,\varepsilon) \\ (-1,\varepsilon) & (0,1) & (0,0) & (+1,\varepsilon) \end{array}\right].$$

Note that M_1 specifies both possible schedules two times. However, this redundancy disappears in the inductive step.

Inductive step from $h \geq 1$ to $h + 1$: Note that, for $\ell \in \{0, 1\}$, restricting $\mathcal{D}^{h+1}$ to events of the form (s, x') where $x' = \ell\, x''$ leads to a partial order isomorphic to $\mathcal{D}^h$; these two partial orders are disjoint, and we denote them by $\mathcal{D}^h(\ell)$, $\ell \in \{0, 1\}$; in fact, $\mathcal{D}^h(0) \cup \mathcal{D}^h(1) \cup \{(-1, \varepsilon), (+1, \varepsilon)\}$ forms a partition of $\mathcal{D}^{h+1}$. We assume that the matrix M_h is known (the inductive hypothesis); for $\ell \in \{0, 1\}$, we denote its image under the (entry-wise) mapping $(s, x') \mapsto (s, \ell\, x')$ by $M_h(\ell)$. In other words, $M_h(\ell)$ is the matrix that defines our (soon proved to be 3-hitting) family of schedules for $\mathcal{D}^h(\ell)$; we will also apply the same notation to A, B, C, and D.

Finally, we will need two auxiliary schedules for double trees, which we call *left* and *right* traversals. The left traversal λ of $\mathcal{D}^{h+1}$ is defined inductively as follows: it first schedules $(-1, \varepsilon)$, then takes the left traversal of $\mathcal{D}^h(0)$, then the left traversal of $\mathcal{D}^h(1)$, and then schedules $(+1, \varepsilon)$. The right traversal ρ is defined symmetrically. Denote by $\lambda(\ell)$ and $\rho(\ell)$ left and right traversals of $\mathcal{D}^h(\ell)$, respectively (we omit reference to h since this does not create confusion). Then

$$A_{h+1} = \left[\begin{array}{c|c|c} (-1,\varepsilon) & & \\ \vdots & A_h(0) & A_h(1) \\ (-1,\varepsilon) & & \\ \hline (-1,\varepsilon) & \multicolumn{2}{c}{\lambda(0)} \\ \hline (-1,\varepsilon) & \multicolumn{2}{c}{\lambda(1)} \end{array}\right], \quad B_{h+1} = \left[\begin{array}{c|c|c} & & (+1,\varepsilon) \\ B_h(1) & B_h(0) & \vdots \\ & & (+1,\varepsilon) \\ \hline \multicolumn{2}{c|}{\lambda(1)} & (+1,\varepsilon) \\ \hline \multicolumn{2}{c|}{\lambda(0)} & (+1,\varepsilon) \end{array}\right],$$

$$C_{h+1} = \left[\begin{array}{c|c|c} (-1,\varepsilon) & & \\ \vdots & C_h(1) & C_h(0) \\ (-1,\varepsilon) & & \\ \hline (-1,\varepsilon) & \multicolumn{2}{c}{\rho(0)} \\ \hline (-1,\varepsilon) & \multicolumn{2}{c}{\rho(1)} \end{array}\right], \quad D_{h+1} = \left[\begin{array}{c|c|c} & & (+1,\varepsilon) \\ D_h(0) & D_h(1) & \vdots \\ & & (+1,\varepsilon) \\ \hline \multicolumn{2}{c|}{\rho(1)} & (+1,\varepsilon) \\ \hline \multicolumn{2}{c|}{\rho(0)} & (+1,\varepsilon) \end{array}\right].$$

Our result is that, for each h, M_h is a 3-hitting family of schedules for $\mathcal{D}^h$. The key part of the proof relies on the following auxiliary property, which is a stronger form of the 2-hitting condition.

Lemma 9. *For any pair of distinct events $\boldsymbol{a} = (a_1, a_2)$ from $\mathcal{D}^h$, if there is a schedule for $\mathcal{D}^h$ that hits $\boldsymbol{a}$, then each of the matrices $\left[\, A_h | B_h \,\right]$ and $\left[\, C_h | D_h \,\right]$ contains a schedule for $\mathcal{D}^h$ where a_1 is placed in the first half and a_2 is placed in the second half.*

5 d-hitting families for $d \geq 3$ of size $f(d)(\log n)^{d-1}$

Fix some d and let $\mathcal{T}^h$ be a complete binary tree of height h, as defined in subsection 3.1. In this section we prove the following theorem.

Theorem 10. *For any $d \geq 2$ the complete binary tree of height h has a d-hitting family of schedules of size* $\exp(d) \cdot h^{d-1}$.

Note that in terms of the number of nodes of $\mathcal{T}^h$, which is $n = 2^{h+1} - 1$, Theorem 10 gives a d-hitting family of size polylogarithmic in n. The proof of the theorem is constructive, and we divide it into three steps. The precise meaning to the steps relies on auxiliary notions of a *pattern* and of d-tuples *conforming to* a pattern; we give all necessary definitions below.

Lemma 11. *For each admissible d-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ there exists a pattern p such that $\boldsymbol{a}$ conforms to p.*

Lemma 12. *For each pattern p there exists a schedule α_p that hits all d-tuples $\boldsymbol{a}$ that conform to p.*

Lemma 13. *The total number of patterns, up to isomorphism, does not exceed* $\exp(d) \cdot h^{d-1}$.

The statement of Theorem 10 follows easily from these lemmas. The key insight is the definition of the pattern and the construction of Lemma 12.

In the sequel, for partial orders that are trees directed from the root we will use the standard terminology for graphs and trees (relying on Hasse diagrams): node, outdegree, siblings, 0- and 1-principal subtree of a node, isomorphism. We denote the parent of a node u by $\operatorname{par} u$ and the *least common ancestor* of nodes u and v by $\operatorname{lca}(u, v)$.

If T is a tree and $X \subseteq T$ is a subset of its nodes, then by $[X]$ we denote the lca-closure of X: the smallest set $Y \subseteq T$ such that, first, $X \subseteq Y$ and, second, for any $y_1, y_2 \in Y$ it holds that $\operatorname{lca}(y_1, y_2) \in Y$. The following claim is a variation of a folklore Lemma 1 in [7].

Claim 14. $|[X]| \leq 2\,|X| - 1$.

Definition 15 (pattern). A *pattern* is a quintuple $p = (D, \preccurlyeq, s, \ell, \pi)$ where:

— $d \leq |D| \leq 2d - 1$,
— $(D, \preccurlyeq)$ is a partial order which is, moreover, a tree directed from the root,
— the number of non-leaf nodes in $(D, \preccurlyeq)$ does not exceed $d - 1$,
— each node of $(D, \preccurlyeq)$ has outdegree at most 2,
— the partial function $s\colon D \rightharpoonup \{0, 1\}$ specifies, for each pair of siblings v_1, v_2 in $(D, \preccurlyeq)$, which is the left and which is the right child of its parent: $s(v_t) = 0$ and $s(v_{3-t}) = 1$ for some $t \in \{1, 2\}$; the value of s is undefined on all other nodes of D,
— the partial function $\ell\colon D \rightharpoonup \{0, 1, \ldots, h - 1\}$ associates a *layer* with each non-leaf node of $(D, \preccurlyeq)$, so that $u \prec v$ implies $\ell(u) < \ell(v)$; the value of ℓ is undefined on all leaves of D, and
— π is a schedule for $(D, \preccurlyeq)$.

We remind the reader that the symbol $\leq$ refers to the same partial order as $\mathcal{T}^h$.

Definition 16 (conformance). Take any pattern $p = (D, \preccurlyeq, s, \ell, \pi)$ and any tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ of d distinct elements of the partial order $\mathcal{T}^h$. Consider the set $\{a_1, \ldots, a_d\}$: the restriction of $\leq$ to its lca-closure $A = [\{a_1, \ldots, a_d\}]$ is a binary tree, $(A, \leq)$. Suppose that the following conditions are satisfied:

a) the trees $(D, \preccurlyeq)$ and $(A, \leq)$ are isomorphic: there exists a bijective mapping $i\colon D \to A$ such that $v_1 \preccurlyeq v_2$ in D iff $i(v_1) \leq i(v_2)$ in $\mathcal{T}^h$;
b) the partial function s correctly indicates left- and right-subtree relations: for any $v \in D$, $s(v) = b \in \{0, 1\}$ if and only if $i(v)$ lies in the b-principal subtree of $i(\operatorname{par}(v))$;
c) the partial function ℓ correctly specifies the layer inside $\mathcal{T}^h$: for any non-leaf $v \in D$, $\ell(v) = |i(v)|$; recall that elements of $\mathcal{T}^h$ are binary strings from $\{0, 1\}^{\leq h}$;
d) the schedule π for $(D, \preccurlyeq)$ hits the tuple $i^{-1}(\boldsymbol{a}) = (i^{-1}(a_1), \ldots, i^{-1}(a_d))$.

Then we shall say that the tuple $\boldsymbol{a}$ *conforms to* the pattern p.

```
<img src="..." onload="javascript:loaded()"/>
<script>
  function loaded() {
    document.getElementById('p').innerHTML = 'Loaded';
  }
</script>
<p id="p">Waiting...</p>
```

Fig. 2. Example of bugs of depth $d = 2$ and $d = 3$ in a web page

We now sketch the proof of Lemma 12. Fix any pattern $p = (D, \preccurlyeq, s, \ell, \pi)$. Recall that we need to find a schedule α_p that hits all d-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to p. We will pursue the following strategy. We will cut the tree $\mathcal{T}^h$ into multiple pieces; this cutting will be entirely determined by the pattern p, independent of any individual $\boldsymbol{a}$. Each piece in the cutting will be associated with some element $c \in D$, so that each element of D can have several pieces associated with it. In fact, every piece will form a subtree of $\mathcal{T}^h$ (although this will be of little importance). The key property is that, for every d-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to p, if i is the isomorphism from Definition 16, then each event a_k, $1 \le k \le d$, will belong to a piece associated with $i^{-1}(a_k)$. As a result, the desired schedule α_p can be obtained in the following way: arrange the pieces according to how π schedules elements of D and pick any possible schedule inside each piece. This schedule will be guaranteed to meet the requirements of the lemma.

6 From hitting families to systematic testing

Hitting families of schedules serve as a theoretical framework for systematically exposing all bugs of small depth. However, bridging the gap from theory to practice poses several open challenges, which we describe in this section.

To make the discussion concrete, we focus on a specific scenario: testing the rendering of web pages in the browser. Web pages exhibit event-driven concurrency: as the browser parses the page, it concurrently executes JavaScript code registered to handle various automatic or user-triggered events. Many bugs occur as a consequence of JavaScript's ability to manipulate the structure of the page while the page is being parsed. Previous work shows such bugs are often of small depth [10,17].

As an example, consider the web page in Fig. 2. In the example, the image (represented by the `<img>` tag) has an on-load event handler that calls the function `loaded()` once the image is loaded. The function, defined in a separate script block, changes the text of the paragraph `p` to *Loaded.* There are two potential bugs in this example. The first one is of depth $d = 2$, and it occurs if the image is loaded quickly (for example, from the cache), before the browser parses the `<script>` tag. In this case, the on-load handler tries to call an undefined function. The second bug is of depth $d = 3$, and it occurs if the handler is

```
<img src="..." onload="javascript:loaded()"/>
<script>
  function loaded() {
    var p = document.getElementById('p');
    if (p == null) {
      setTimeout(loaded, 10);
    } else {
      p.innerHTML = 'Loaded';
    }
  }
</script>
<p id="p">Waiting...</p>
```

Fig. 3. Using a timer to fix the bug from Fig. 2 involving a non-existent element

executed after the `<script>` tag is parsed, but before the `<p>` tag is parsed. In this case, the function `loaded()` tries to access a non-existent HTML element.

Next, we identify and discuss three challenges.

Events and partial orders need not be static. Our theoretical model assumes a static partially-ordered set of events, and allows arbitrary reordering of independent (incomparable) events. For the web page in Fig. 2, there are three parsing events (corresponding to the three HTML tags) and an on-load event. The parsing events are chained in the order their tags appear in the code. The on-load event happens after the `<img>` tag is parsed, but independently of the other parsing events, giving a tree-shaped partial order.

In more complex web pages, the situation is not so simple. Events may be executions of scripts with complex internal control-flow and data dependencies, as well as with effect on the global state. Once a schedule is reordered, new events might appear, and some events might never trigger. An example showing a more realistic situation is given in Fig. 3. In order to fix the bug involving a non-existent HTML element `p`, the programmer now explicitly checks the result of `getElementById()`. If `p` does not exist (`p == null`), the programmer sets a timer to invoke the function `loaded()` again after 10 milliseconds. As a consequence, depending on what happens first—the on-load event or the parsing of `<p>`—we may or may not observe one or more timeout events. Note that the chain of timeout events also depends on parsing the `<script>` tag. If the tag is not parsed, the `loaded()` function does not exist, so no timer is ever set. Moreover, the number of timeout events depends on when exactly the `<p>` tag is parsed.

The example shows that there is a mismatch between the assumption of static partially ordered events and the dynamic nature of events occuring in complex web pages. Ideally, the mismatch should be settled in future work by explicitly modeling this dynamic nature. However, even the current theory of hitting families can be applied as a testing heuristic. While we lose completeness (in the sense of hitting all depth-d bugs), we retain the variety of different event orderings. In the context of web pages, an initial execution of a page gives us an initial partially ordered set of events. We use it to construct a hitting family of schedules, which we optimistically try to execute. The approach is based on the notion of *approximate replay*, which is employed by R^4, a stateless model checker for web pages [10]. We come back to this approach later in the section.

Another approach is to construct hitting families *on the fly*: Such a construction would unravel events and the partial order dynamically during execution, and non-deterministically construct a schedule from a corresponding hitting family. In this way, the issue of reordering events in an infeasible way does not arise, simply because nothing is reordered. This is in line with how PCT [3] and delay-bounded scheduling [6] work. On-the-fly constructions of small hitting families are a topic for future work.

Beyond trees. Our results on trees are motivated by the existing theoretical models of asynchronous programs [11,8,6], where the partial order induced by event handlers indeed form trees. However, in the context of web pages, events need not necessarily be ordered as nodes of a tree. An example of a feature that introduces additional ordering constraints is deferred scripts. Scripts marked as deferred are executed after the page has been loaded, and they need to be executed in the order in which their corresponding `<script>` tags were parsed [15]. The tree approximation corresponds to testing the behavior of pages when the deferred scripts are treated as normal scripts and loaded right away. An open question is to generalize our construction to other special cases of partial orders that capture common programming idioms.

Unbalanced trees. For a tree of height h, constructions from Sections 4 and 5 give 3-hitting families of size $O(h)$ and $O(h^2)$, respectively. If the tree is balanced, the cardinality of these families are exponentially smaller than the number of events in the tree. However, in the web page setting, trees are not balanced.

In order to inspect the shape of partial orders occurring in web pages, we randomly selected 24 websites of companies listed among the top 100 of Fortune 500 companies. For each website, we used R^4 [10] to record an execution and construct the happens-before relation (the partial order). Table 1 shows the number of events and the height of the happens-before graph for the websites. The results indicate that a typical website has most of the events concentrated in a backbone of very large height, proportional to the total number of events.

The theory shows that going below $\Theta(h)$ is impossible in this case unless $d < 3$; and this can indeed lead to large hitting families: for example, our construction for $h = 1000$ and $d = 4$ corresponds to several million tests. However, not all schedules of the partial ordering induced by the event handlers may be relevant: if two events are independent (commute), one need not consider schedules which only differ in their ordering. Therefore, since hitting families are defined on an *arbitrary* partial order, not only on the happens-before order, we can use

Table 1. For each website, the table show the number of events in the initial execution, the height of the partial order (happens-before graph), the number of schedules generated for $d = 3$, and the number of schedules for $d = 3$ with pruning based on races.

Website	# Events	Height	$d = 3$	$d = 3$ (pruned)
abc.xyz	337	288	561	0
newscorp.com	1362	875	2689	100
thehartford.com	2018	1547	3913	138
www.allstate.com	4534	3822	9023	106
www.americanexpress.com	2971	2586	5897	340
www.bankofamerica.com	2305	2095	4561	150
www.bestbuy.com	301	248	576	10
www.comcast.com	188	118	337	16
www.conocophillips.com	4184	3478	8286	248
www.costco.com	7331	6390	14614	364
www.deere.com	2286	1902	4516	236
www.generaldynamics.com	2820	2010	5611	272
www.gm.com	2337	1473	4600	94
www.gofurther.com	1117	638	2154	568
www.homedepot.com	3780	2100	7515	1526
www.humana.com	5611	4325	11174	2058
www.johnsoncontrols.com	2953	2395	5881	450
www.jpmorganchase.com	4134	3519	8247	1316
www.libertymutual.com	3885	3560	7735	324
www.lowes.com	6938	4383	13778	3438
www.massmutual.com	3882	3313	7682	1852
www.morganstanley.com	2752	2301	5402	128
www.utc.com	4081	3266	8100	206
www.valero.com	2116	1849	4178	38

additional information, such as (non-)interference of handlers, to reduce the partial ordering first.

For web pages, we apply a simple partial order reduction to reduce the size of the input trees in the following way. We say a pair of events *race* if they both access some memory location or some DOM element, with at least one of them writing to this location or the DOM element. Events that do not participate in races commute with all other events, so they need not be reordered if our goal is to expose bugs.

R^4 internally uses a race detection tool (EventRacer [17]) to over-approximate the set of racing events. In order to compute hitting families, we construct a pruned partial order from the original tree of events. As an example, for $d = 3$ and the simple $O(n^{d-2})$ construction, instead of selecting a_1 arbitrarily, we select it from the events that participate in races. We then perform the left-to-right and right-to-left traversals as usual. In total, the number of generated schedules is $2r$, where r is the number of events participating in races. This number can be significantly smaller than $2n$, as can be seen in the fourth ($d = 3$) and fifth ($d = 3$ pruned) columns of Table 1.

7 Conclusions

We have introduced hitting families as the basis for systematic testing of concurrent systems and studied the size of optimal d-hitting families for trees and related partial orders.

We have shown that a range of combinatorial techniques can be used to construct d-hitting families: we use a greedy approach, a randomized approach, and a construction based on DFS traversals; we also develop a direct inductive construction and a construction based on what we call patterns. The number of schedules in the pattern-based construction is polynomial in the height—for balanced trees, this is exponentially smaller than the total number of nodes.

Our development of hitting families was motivated by the testing of asynchronous programs, and we studied the partial ordering induced by the happens-before relationship on event handlers. While this ordering gives a useful testing heuristic in scenarios such as rendering of web pages, the notion of hitting families applies to any partial ordering, and we leave its further uses to future work.

Acknowledgements. We thank Madan Musuvathi for insightful discussions and comments.

References

1. Noga Alon and Joel H. Spencer. *The Probabilistic Method.* Wiley, 2008. 3rd edition.
2. Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Precedence constraint scheduling and connections to dimension theory of partial orders. *Bulletin of the EATCS*, 95:37–58, 2008.
3. Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, and Santosh Nagarakatte. A randomized scheduler with probabilistic guarantees of finding bugs. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2010, Pittsburgh, Pennsylvania, USA, March 13-17, 2010*, pages 167–178, 2010.
4. Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1557–1576. SIAM, 2013.
5. Ben Dushnik and E. W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610, 1941.
6. Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422, 2011.

7. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012.
8. Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6, 2012.
9. Rajneesh Hegde and Kamal Jain. The hardness of approximating poset dimension. *Electronic Notes in Discrete Mathematics*, 29:435–443, 2007.
10. Casper Svenning Jensen, Anders Møller, Veselin Raychev, Dimitar Dimitrov, and Martin T. Vechev. Stateless model checking of event-driven applications. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 57–73, 2015.
11. Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In *POPL '07: Proc. 34th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, pages 339–350. ACM Press, 2007.
12. László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.
13. Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2008, Seattle, WA, USA, March 1-5, 2008*, pages 329–339, 2008.
14. Madan Musuvathi and Shaz Qadeer. CHESS: systematic stress testing of concurrent software. In *Logic-Based Program Synthesis and Transformation, 16th International Symposium, LOPSTR 2006, Venice, Italy, July 12-14, 2006, Revised Selected Papers*, pages 15–16, 2006.
15. Boris Petrov, Martin T. Vechev, Manu Sridharan, and Julian Dolby. Race detection for web applications. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, pages 251–262, 2012.
16. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 93–107, 2005.
17. Veselin Raychev, Martin T. Vechev, and Manu Sridharan. Effective race detection for event-driven programs. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 151–166, 2013.
18. Alexander Sapozhenko. On the complexity of disjunctive normal forms obtained with a gradient algorithm. In *Diskretnyj Analiz (Discrete Analysis)*, volume 21, pages 62–71. Institute for Mathematics in the Siberian Section of the Academy of Sciences, Novosibirsk, 1972. In Russian.
19. Bernd S.W. Schröder. *Ordered Sets: An Introduction.* Springer, 2003.
20. Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV'06: Proc. 18th Int. Conf. on Computer Aided Verification*, volume 4144 of *LNCS*, pages 300–314. Springer, 2006.

21. Sherman K. Stein. Two combinatorial covering theorems. *J. Comb. Theory, Ser. A*, 16(3):391–397, 1974.
22. William T. Trotter. A generalization of Hiraguchi's: Inequality for posets. *J. Comb. Theory, Ser. A*, 20(1):114–123, 1976.
23. William T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.
24. Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.