Learning Union of Integer Hypercubes with Queries
(with Applications to Monadic Decomposition)

Oliver Markgraf[1](✉), Daniel Stan[1], and Anthony W. Lin[1,2]

[1] TU Kaiserslautern, Kaiserslautern, Germany
{markgraf,stan,lin}@cs.uni-kl.de
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract. We study the problem of learning a finite union of integer (axis-aligned) hypercubes over the d-dimensional integer lattice, i.e., whose edges are parallel to the coordinate axes. This is a natural generalization of the classic problem in the computational learning theory of learning rectangles. We provide a learning algorithm with access to a minimally adequate teacher (i.e. membership and equivalence oracles) that solves this problem in polynomial-time, for any fixed dimension d. Over a non-fixed dimension, the problem subsumes the problem of learning DNF boolean formulas, a central open problem in the field. We have also provided extensions to handle infinite hypercubes in the union, as well as showing how subset queries could improve the performance of the learning algorithm in practice. Our problem has a natural application to the problem of monadic decomposition of quantifier-free integer linear arithmetic formulas, which has been actively studied in recent years. In particular, a finite union of integer hypercubes correspond to a finite disjunction of monadic predicates over integer linear arithmetic (without modulo constraints). Our experiments suggest that our learning algorithms substantially outperform the existing algorithms.

1 Introduction

Suppose that we are interested in finding a formula $\varphi(\bar{x})$ over some theory T (e.g. integer linear arithmetic) to "capture" a certain phenomenon, which in verification could be, for instance, an invariant that a program satisfies some safety property. The process of discovering φ can be captured by the notion of a *learning algorithm* by allowing certain types of queries as an interface to some teacher [3]. Most standard learning frameworks can be captured in this way. Here are some examples. Valiant's well-known notion of *PAC-learning* can be captured by an oracle that returns a new random sample from an unknown distribution. Angluin's well-known notion of *exact learning* [2,3] can be captured by an interaction with the so-called *minimally adequate teachers*, which

This research was supported by the ERC Starting Grant 759969 (AV-SMP) and Max-Planck Fellowship.

can answer membership and equivalence queries. This has many applications in verification, e.g., verification of parameterized systems [10,20,23] and compositional verification [9]. Another learning framework that has become very popular in verification is CEGIS (Counterexample Guided Inductive Synthesis) [21,27], wherein a learning algorithm can ask equivalence queries, but expect various types of "constraint-like" counterexamples (e.g. implication counterexamples) to be returned by the teacher. This is of course in contrast to Angluin's exact learning setting, wherein the teacher may return only a positive/negative counterexample (a point in the symmetric difference of the target concept and the hypothesis).

In this paper, we study the problem of learning sets of points over the d-dimensional integer lattice that can be expressed as a *finite union of integer (axis-aligned, a.k.a. rectilinear) hypercubes*, i.e., whose edges are parallel to the coordinate axes. Such a concept class of course forms a strict subclass of sets of points that are definable by a formula $\varphi(x_1, \ldots, x_d)$ in the integer linear arithmetic (a.k.a. *semilinear sets*), which have been addressed in several papers including [1,17,28], whose PAC-learnability is as hard as PAC-learning boolean formulas in DNF [16]—a long-standing open problem in learning theory—when binary representations are permitted (even over dimension one [1]). That said, finite unions of integer hypercubes are a concept class that naturally arises in computer science. Below we mention a few examples.

The problem of learning rectangles (2-cube) and generalization to d-dimension are a classic example in computational learning theory, e.g., see [16,22]. Maass and Turán [22] showed for example that the d-dimensional rectilinear cubes can be learned in polynomial-time with $O(\log n)$ queries, where the corners of the cubes are represented in binary. The authors posed as an open problem if one can learn a union of two (possibly overlapping) rectangles with only $O(\log n)$ equivalence queries. Chen [11] showed that this can be learned with 2 equivalence queries and $O(d. \log n)$ membership queries. Later Chen and Ameur [12] showed that there is a polynomial-time algorithm using at most $O(\log^2 n)$ queries. The same paper left as an open problem if there is a polynomial-time exact learning algorithm that learns finite unions of rectilinear cubes over a fixed dimension d. In this paper, *we answer this in the positive*, and further show that this can be extended to allow *infinite rectilinear hypercubes*, which in turn allow interesting applications in formal verification, as we discuss below.

Finite unions of rectilinear cubes arise naturally in program analysis and verification. Here we mention two examples. First, solving games over a large game graph has benefited from constraint-based approaches, where winning regions can be succinctly represented and checked efficiently [6]. For example, the discretization of the Cinderella-Stepmother problem [6] admits winning regions that may be represented by a union of a small number of cubes. Secondly, verification algorithms benefit from optimization techniques like monadic decomposition [29], where the aim is the rewriting of a given quantifier-free SMT formula $\varphi(x_1, \ldots, x_n)$ into an equivalent boolean combination of monadic predicates $\psi(x_i)$ in some special form, i.e., typically in DNF [5,7,15,19], or by an if-then-else formula [29], which could sometimes be exponentially smaller than the DNF equivalent representation. Veanes *et al.* [29] provided a generic semi-decision procedure for performing this monadic decomposition as an if-then-else formula, which works regardless of the base theory. The restriction of the problem to the quantifier-free theory of integer linear arithmetic (with and without extra modulo constraints) was studied in [15], wherein the problem was shown to be coNP-complete and a monadic decomposition could be exponentially large in general. For the subcase without modulo constraints, a monadic decomposition in DNF corresponds precisely to a finite union of (possibly infinite) rectilinear hypercubes, which is the subject of this paper. We describe below how oracles for memberships and equivalence (as well as more powerful queries like subsets) admit a fast implementation via an SMT-solver, which enable our learning algorithms to be applied to compute such a monadic decomposition.

Contributions. We study the problem of learning finite unions of rectilinear hypercubes (over $\mathbb{Z}^d$) in Angluin's exact learning framework with membership and equivalence queries [2,3]. Our result is a polynomial-time exact learning algorithm for learning finite unions of rectilinear hypercubes over $\mathbb{Z}^d$ for fixed d. This answers an open problem of [12]. As observed in [12], over non-fixed d, this problem generalizes DNF since each term can be seen as a hypercube over $\{0,1\}^d$. That is, without fixing d, the problem is as hard as learning unrestricted DNF, which is well-known to be a major open problem in computational learning theory [4].

In view of applying our learning algorithm to the monadic decomposition problem [15,29] for quantifier-free integer linear arithmetic formulas, we consider two extensions. Firstly, we allow *infinite hypercubes*. For example, over 1-dimension, these would include infinite intervals like $[7,\infty)$, which would correspond to the formula $x \geq 7$. Secondly, we observe that the *subset query* (i.e. checking if the target concept includes a given finite union H of hypercubes) is not an expensive query for performing monadic decomposition, i.e., it would correspond to a single satisfiability check of a quantifier-free integer linear arithmetic formula, which can be handled easily by an SMT-solver. Subset queries belong to one of the standard types of queries in Angluin's active learning framework, e.g., see [3]. For this reason, we provide an optimization of our learning algorithm by means of subset queries.

We implemented these learning algorithms (vanilla and various optimization including subset queries and "unary/binary acceleration"), using Z3 [26] as the backend for answering equivalence and subset queries (each a satisfiability check of a quantifier-free formula). We have performed a micro-benchmarking to stress-test our algorithms against the generic monadic decomposition procedure of [29], which also use Z3 as the backend, using various geometric objects over $\mathbb{Z}^d$ as benchmarks. Our experiments suggest that our algorithms substantially outperform the generic procedure.

Organization. Preliminaries are in Sect. 2. We present the *overshooting algorithm* that witnesses polynomial learnability of finite unions of rectilinear cubes over a fixed dimension d with membership and equivalence in Sect. 3. In Sect. 4, we provide two extensions: (1) how subset queries could help speed up the overshooting algorithm, (2) how the algorithm could be extended to handle infinite cubes. Applications to monadic decomposition and experiments are presented in Sect. 5. We conclude in Sect. 6.

We refer the reader to the technical report [25] when proofs are omitted and to the artifact [24] for implementation and benchmark details.

2 Preliminaries

We introduce below some common mathematical notations: $\mathbb{N}$ and $\mathbb{Z}$ are the sets of natural numbers and integers, respectively. For $a, b \in \mathbb{Z}$, we write $[a, b] = \{i \mid a \leq i \leq b\}$; For any set X, we denote its power-set $\mathcal{P}(X)$ and its cardinal $|X| \in \mathbb{N} \uplus \{\infty\}$; Given two sets A, B, the *symmetric difference* is written $A \Delta B = A \backslash B \cup B \backslash A$;

When analyzing complexity of the presented algorithms, we assume binary encoding for any number $n \in \mathbb{Z}$, which is part of the input of the considered algorithms, namely, $\text{size}(n) = 1 + \lceil \log(|n|+1) \rceil$, where log is the base 2 logarithm.

Hypercubes. For a fixed *dimension* $d \in \mathbb{N}$, we consider the *discrete lattice* $\mathbb{Z}^d$. A *point* $\mathbf{v} \in \mathbb{Z}^d$ can be described by its coordinates $\mathbf{v}[k]$ for $k \in [1, d]$. Let $\mathbf{v}[k/\alpha]$ denote the vector $\mathbf{v}$ where the i-th coordinate has been replaced by $\alpha \in \mathbb{Z}$. The notation $\mathbf{0}^d = (0, \ldots, 0) \in \mathbb{Z}^d$ denotes the origin, or simply $\mathbf{0}$ when the dimension is clear from context. We use standard notation for component-wise additions and scalar multiplication. In particular, for $\alpha \in \mathbb{Z}$, $\mathbf{v} + \alpha \cdot \mathbf{v}'$ denotes the vector $\mathbf{v}'' \in \mathbb{Z}^d$ such that for all i, $\mathbf{v}''[i] = \mathbf{v}[i] + \alpha \cdot \mathbf{v}'[i]$. For $1 \leq i \leq d$, we write $\mathbf{e}_i$ for the i-th *elementary vector*, $\mathbf{e}_i = \mathbf{0}[i/1]$. We shall be mostly using the standard *component-wise order* $\leq$ over vectors in $\mathbb{Z}^d$: $\mathbf{v} \leq \mathbf{v}'$ iff for all i, $\mathbf{v}[i] \leq \mathbf{v}'[i]$. We finally denote the size of a vector as the sum of the sizes of its components: $\text{size}(\mathbf{v}) = \sum_{i=1}^{d} \text{size}(\mathbf{v}[i])$, for any $\mathbf{v} \in \mathbb{Z}^d$.

Our main study focuses on *rectilinear hypercubes* (*cubes* for short), i.e., any set of points of the form $C = \{\mathbf{v} \mid \underline{\mathbf{v}} \leq \mathbf{v} \leq \overline{\mathbf{v}}\}$ for some $\underline{\mathbf{v}} \leq \overline{\mathbf{v}} \in \mathbb{Z}^d$. The size of C is uniquely defined as $\text{size}(C) = \text{size}(\underline{\mathbf{v}}) + \text{size}(\overline{\mathbf{v}})$. On the contrary, an arbitrary finite set X has no unique representation as a finite union of cubes, therefore we define its size as the size of its best representation:

$$\text{size}(X) = \min \left\{ \sum_{i=1}^{n} \text{size}(\underline{\mathbf{v}}_i) + \text{size}(\overline{\mathbf{v}}_i) \;\middle|\; \exists n, \underline{\mathbf{v}}_1 \ldots \overline{\mathbf{v}}_n : X = \bigcup_{i=1}^{n} \text{Cube}(\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i) \right\}$$

We adopt here a worst-case analysis approach, where our later reasoning and complexity analysis are valid for any representation, they are in particular valid for its best representation.

Learning Model. We first recall some standard definition from computational learning theory; for more, see [16]. Fix a countable *base set* $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$, where the sets $\mathcal{D}_i$'s are pairwise disjoint. The problem of learning boolean formulas in DNF uses $\mathcal{D}_i = \{0,1\}^i$, i.e., the set of all binary sequences of length i, which can be thought of as a set of all assignments to a boolean function over $x_1, \ldots, x_i$. The learning problem in this paper uses $\mathcal{D}_i = \mathbb{Z}^i$. A *concept* X is simply a subset of $\mathcal{D}_i$, for some $i \in \mathbb{Z}_{>0}$. For example, when $\mathcal{D}_i = \{0,1\}^i$, a concept is simply a boolean function over $x_1, \ldots, x_i$. When we speak of a learning problem, we always have a fixed set of representations in mind. For example, when we speak of learning boolean formulas in DNF (Disjunctive Normal Form), the representation φ_X of a boolean function X has to be a formula over $x_1, \ldots, x_i$ in DNF. For example, X could be a boolean function, whereas φ_X a DNF formula representing X. Note that a concept could admit many possible representations. A *concept class* $\mathcal{C} = \bigcup_{i=1}^{\infty} \mathcal{C}_i$ is a set of concepts, where $\mathcal{C}_i \subseteq \mathcal{P}(\mathcal{D}_i)$. For example, $\mathcal{C}_i$ could be the set of boolean functions over variables $x_1, \ldots, x_i$. When the set of representations for $\mathcal{C}$ is fixed (e.g. DNF for representing boolean functions), we could define size(X) of the concept X to be the size of the smallest representation of X. In this paper, we are dealing with the concept class $\mathcal{C}_d \subseteq \mathcal{P}(\mathbb{Z}^d)$ of sets of integer points that can be represented as a finite union of rectilinear hypercubes over $\mathbb{Z}^d$. Earlier in this section we have defined this concept, as well as the size of the representation. To avoid notational clutter, we will often denote the concept class $\mathcal{C}_d$ by $\mathcal{C}$ because our algorithm typically assumes that d is fixed.

In Angluin's active learning framework [2,3], the learner has access to oracles (a.k.a. teachers) that could provide hints about the target concept X to the learner. A *minimally adequate teacher* must be able to answer membership and equivalence queries.

Definition 1 (M+EQ Oracles). *Consider some target concept $X \in \mathcal{C}_d$ for some concept class $\mathcal{C} = \bigcup_{d=1}^{\infty} \mathcal{C}_d$ and let $\bot, \top \notin \mathcal{D}$ be two fresh symbols.*

- *A* membership oracle *(M) for X is a function $\Phi_X : \mathcal{D}_d \to \{\top, \bot\}$, which outputs $\top$ iff $\mathbf{v} \in X$.*
- *An* equivalence oracle *(EQ) for X is a function $\Psi_X : \mathcal{C}_d \to \mathcal{D}_d \uplus \{\top\}$ such that for all* hypothesis *$H \in \mathcal{C}$, $\Psi_X(H) \in (H \Delta X) \uplus \{\top\}$ and $\Psi_X(H) = \top$ implies $H = X$.*

Intuitively, an equivalence oracle tells, for any hypothesis $H \in \mathcal{C}$, whether $H = X$. If yes, $\top$ is returned; if not, it provides a *counterexample*, namely a point in the symmetric difference. Angluin has considered other types of queries as well in her framework including subset/superset queries and difference queries (e.g. see her excellent survey [3]). We will use the subset queries in Sect. 4.

A learning algorithm $\mathcal{A}$ is said to *learn* the concept class $\mathcal{C} = \bigcup_{d=1}^{\infty} \mathcal{C}_d$ if, given d as input and any unknown target concept X, it terminates and outputs a representation of X after a finite amount of interaction with the oracles. Assuming that the oracle always returns the shortest counterexamples, its running time is defined to be number of steps (measured in d and size(X)) that $\mathcal{A}$ takes to output a representation of X. The complexity *comp*$(d, \text{size}(X))$ of $\mathcal{A}$ measures the number of steps taken in the worst case for all d and size(X). It runs in polynomial time if *comp* is a polynomial function. It remains a long-standing open problem in computational learning theory if there is a learning algorithm for boolean formulas represented in DNF, which is true for almost all major models including exact learning and PAC (see [4]). Over geometric concepts including hypercubes and semilinear sets, the dimension d is sometimes considered a fixed parameter, e.g., see [1,12,17,22].

3 Minimally Adequate Teacher

We restrict first our attention to the minimally adequate teacher setting where only a membership and equivalence oracle are provided, and provide constructions for intermediate procedures that can be interpreted as oracles.

3.1 Corner Oracle

At the heart of our learning algorithm is the concept of corners:

Definition 2. *Given a set of points $X \subseteq \mathbb{Z}^d$, a* maximal corner *(resp* minimal corner*) of X is a point $\mathbf{v} \in X$ maximal (resp minimal) with respect to component-wise ordering $\leq$. We write $\overline{\text{Corners}}(X)$ and $\underline{\text{Corners}}(X)$ for the sets of maximal and minimal corners, respectively, and write* $\text{Corners}(X) = \overline{\text{Corners}}(X) \cup \underline{\text{Corners}}(X)$.

Given a membership oracle for some $X \in \mathcal{C}$ containing $\mathbf{0}$, Algorithm 1 returns *some* maximal corner of a given finite subset. Intuitively, for each coordinate i, a binary search is made until a border of X is eventually found. More precisely, we provide the following complexity analysis.

Algorithm 1. Binary search for a maximal corner, assuming $\mathbf{0} \in X$

```
Ensure: Returned value is a maximal corner of $X$
Require: $\mathbf{0} \in X$; $\Phi_X$ a membership oracle for $X$
  function findMaxCorner($\Phi_X$)
      $i \leftarrow 0$;   $\mathbf{v} = \mathbf{0}$
      while $i < d$ do
          $i \leftarrow i + 1$;   $k \leftarrow 1$;   $l \leftarrow 1$;
          if $\Phi_X(\mathbf{v} + \mathbf{e}_i)$ then
              while $\Phi_X(\mathbf{v} + k \cdot \mathbf{e}_i)$ do
                  $l \leftarrow k$;   $k \leftarrow 2k$
              while $k - l > 1$ do
                  if $\Phi_X(\mathbf{v} + \lfloor (k+l)/2 \rfloor \cdot e_i)$ then
                      $l \leftarrow \lfloor (k+l)/2 \rfloor$
                  else
                      $k \leftarrow \lfloor (k+l)/2 \rfloor$
              $\mathbf{v} \leftarrow \mathbf{v} + l \cdot \mathbf{e}_i$;   $i \leftarrow 0$
      return $\mathbf{v}$
```

Proposition 1. *Let Φ_X be a membership oracle for $X = \cup_{i=1}^{n} \mathrm{Cube}(\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i)$ and assume $\mathbf{0} \in X$. Then* findMaxCorner(Φ_X) *terminates after $O\left(\sum_{j=1}^{n} \mathrm{size}(\overline{\mathbf{v}}_j)\right)$ queries and returns some $\overline{\mathbf{v}} \in \overline{\mathrm{Corners}}(X)$.*

This algorithm provides a partial implementation of the following oracle:

Definition 3. *Given $X \in \mathcal{C}$, a* corner oracle *for X is any function $\Theta_X : X \rightarrow \underline{\mathrm{Corners}}(X) \times \overline{\mathrm{Corners}}(X)$.*

A complete implementation of this oracle is provided by noticing that membership oracles can easily be composed:

Remark 1. Assume Φ_A and Φ_B are two given membership oracles, respectively for two arbitrary sets A and B, and $f : \mathbb{Z}^d \rightarrow \mathbb{Z}^d$. One can build membership oracles for $A \cup B$, $A \cap B$, $A \Delta B$, $A \backslash B$ and $f(A)$. In particular:

- By instantiating $f : \mathbf{v} \mapsto -\mathbf{v}$, the previous procedure applied on $\Phi_{f(A)}$ returns some $\mathbf{v} \in \overline{\mathrm{Corners}}(-A)$, so $-\mathbf{v} \in \underline{\mathrm{Corners}}(A)$.
- For any $\mathbf{v}_0 \in A$ and $f : \mathbf{v} \mapsto \mathbf{v} - \mathbf{v}_0$, $\Phi_{f(A)}$ is a membership oracle for $A - \mathbf{v}_0 = \{\mathbf{v} \mid \mathbf{v} + \mathbf{v}_0 \in A\}$ containing $\mathbf{0}$, so findMaxCorner($\Phi_{f(A)}$) returns some $\mathbf{v} \in \overline{\mathrm{Corners}}(A - \mathbf{v}_0)$ so $\mathbf{v} + \mathbf{v}_0 \in \overline{\mathrm{Corners}}(A)$.

In both cases, notice that $\mathrm{size}(f(A)) \leq \mathrm{size}(A) + \mathrm{size}(\mathbf{v}_0) \leq 2\mathrm{size}(A)$.

In the sequel we write Φ_C for the membership oracle of any set C obtained by composing sets whose oracles are provided. We also assume having constructed the two procedures findMaxCorner($\mathbf{v}, \Phi_X$) and findMinCorner($\mathbf{v}, \Phi_X$).

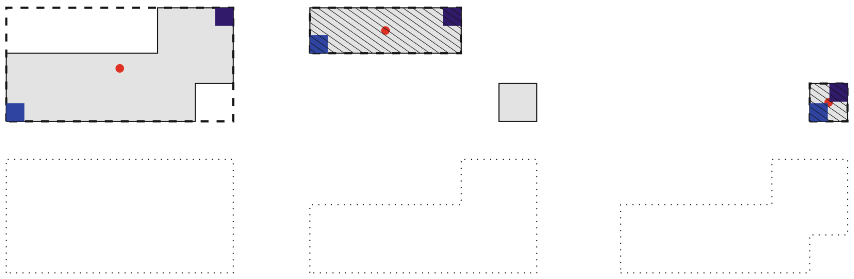
3.2 Overshooting Algorithm

Algorithm 2 Overshooting algorithms

```
Require: $\Phi_X$ membership oracle for $X$, $\Psi_X$ equivalence oracle for $X$
  function LearnCubes($\Phi_X$,$\Psi_X$)
      $H \leftarrow \emptyset$
      repeat
          $\mathbf{v} \leftarrow \Psi_X(H)$
          $H \leftarrow$ Refine($H, \mathbf{v}, \Phi_X$)
      until $\mathbf{v} = \top$
  function RefineSym($H, \mathbf{v}, \Phi_X$)
      $\underline{\mathbf{v}} \leftarrow$ findMinCorner($\mathbf{v}, \Phi_{X \Delta H}$)
      $\overline{\mathbf{v}} \leftarrow$ findMaxCorner($\mathbf{v}, \Phi_{X \Delta H}$)
      return $H \Delta \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$

  function RefineAddRemove($H, \mathbf{v}, \Phi_X$)
      if $\Phi_X(\mathbf{v})$ then
          $\underline{\mathbf{v}} \leftarrow$ findMinCorner($\mathbf{v}, \Phi_{X \backslash H}$)
          $\overline{\mathbf{v}} \leftarrow$ findMaxCorner($\mathbf{v}, \Phi_{X \backslash H}$)
          return $H \cup \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$
      else
          $\underline{\mathbf{v}} \leftarrow$ findMinCorner($\mathbf{v}, \Phi_{H \backslash X}$)
          $\overline{\mathbf{v}} \leftarrow$ findMaxCorner($\mathbf{v}, \Phi_{H \backslash X}$)
          return $H \backslash \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$
```

The core loop of the learning algorithm is presented in the LearnCubes function of Algorithm 2. The hypothesis is initially empty, and is later refined, as long as a counterexample is returned. How to refine the hypothesis given a counterexample? Two implementations of Refine are provided namely RefineSym and RefineAddRemove, giving rise to two variants of the algorithm. In both cases, the refinement takes a counterexample as an input and uses the corner oracle to build a cube C. In the former variant, a symmetric difference between the current hypothesis and C is made, while in the latter, C is either added or removed from the hypothesis.



(a) Step 1: add (b) Step 2: remove (c) Step 3: remove

Fig. 1. Possible run of the overshooting algorithm on two cubes in 2 dimensions

An example run of the RefineAddRemove variant is depicted in Fig. 1. While the above diagrams represent the search space used by the corner oracles, the below diagrams depict the resulting hypothesis after refinement. Initially, the hypothesis is empty (not represented) so the search space coincides with the target set X, which can be represented as a union of two overlapping cubes. A counterexample $\mathbf{v} \in X \backslash H$ is therefore returned by the equivalence oracle. As $\mathbf{v} \in X$, the refinement procedure adds some cube by searching the state space $X \backslash H = X$ around $\mathbf{v}$. A too large cube is then added to the hypothesis, and a negative counterexample $\mathbf{v} \in H \backslash X$ is then returned. The search space is now $H \backslash X$ and the algorithm aims at removing some smaller cube from the hypothesis. After two removals, the final hypothesis coincides with the target.

Hypothesis Representation. Both variants are operating on the hypothesis by applying boolean operations. One can naturally wonder if hypothesis represented by union, symmetric differences and differences of cubes can be handled by oracles operating on the concept class of finite cubes. As a matter of fact, we will observe that $H \Delta X$, $H \backslash X$ and $X \backslash H$ can all be represented in $\mathcal{C}$:

Lemma 1 (Cube intersection and subtraction). *Let* $C_1 = \mathrm{Cube}(\underline{\mathbf{v}}_1, \overline{\mathbf{v}}_1)$ *and* $C_2 = \mathrm{Cube}(\underline{\mathbf{v}}_2, \overline{\mathbf{v}}_2)$ *two cubes.*

Then $C_1 \cap C_2$ *is a cube and* $C_2 \backslash C_1$ *can be written as the disjoint union of* $2d$ *cubes. Moreover, these computations are effective in* $2d$ *operations.*

Intuitively, one can think of a cube subtracted by a smaller cube results in a family of cubes, one for each face of the larger cube. There are $2d$ faces for a cube in dimension d.

3.3 Repetition-Free Complexity

In order to analyze the complexity of both variants of the algorithm, we fix a finite target set $X \in \mathcal{C}_d$ and one of its representation as a union of cubes:

$$X = \bigcup_{i=1}^{n} \mathrm{Cube}(\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i)$$

We prove by induction on the iteration step that H can be expressed as a union of cubes, whose corners are aligned on a particular set of points:

Definition 4 (Abstract grid). *For $1 \leq k \leq d$, we define the sets:*

$$\underline{B}_k = \{\overline{\mathbf{v}}_i[k] + 1 \mid 1 \leq i \leq C\} \cup \{\underline{\mathbf{v}}_i[k] \mid 1 \leq i \leq C\}$$
$$\overline{B}_k = \{\overline{\mathbf{v}}_i[k] \mid 1 \leq i \leq C\} \cup \{\underline{\mathbf{v}}_i[k] - 1 \mid 1 \leq i \leq C\}$$

For any $A \subseteq \mathbb{Z}^d$, we write $A \in \mathcal{B}$ whenever is a finite union of cubes of the form $\mathrm{Cube}(\mathbf{v}, \mathbf{v}')$ *such that for all k, $\mathbf{v}[k] \in \underline{B}_k$ and $\mathbf{v}'[k] \in \overline{B}_k$.*

Intuitively, $\underline{B}_k$ (resp $\overline{B}_k$) describes all the possible k-coordinate for minimal corners (resp maximal). A coordinate for a max corner, i.e. a constraint of the form $x_k \leq \alpha$, can become a coordinate for a minimal corner, i.e. a constraint of the form $x_k \geq \alpha + 1$, when taking the complement during a difference operation, and vice versa.

We observe that $\mathcal{B}$ is stable by union, intersection and difference. In particular, the overshooting algorithms maintain $H \in \mathcal{B}$, namely the hypothesis always has minimal (resp maximal) corners that align with $\underline{B}_k$ (resp $\overline{B}_k$) on the k-th coordinate. Figure 2 provides an example of such points for a target made of the union of two cubes.

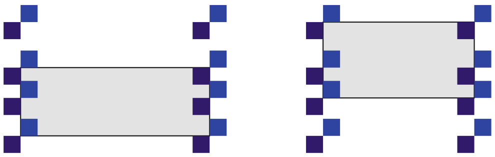


Fig. 2. Possible minimal and maximal corners for cubes appearing in the hypothesis, for a given target space

Since the sets $\underline{B}_k$ and $\overline{B}_k$ are of size at most $2n$ for every k, there are at most $(2n)^{2d}$ possible cubes, polynomial for a fixed d. Assuming $H \in \mathcal{B}$, we can ensure that Lemma 1 maintains a polynomial representation of the hypothesis throughout the algorithm until termination.

Although $\mathcal{B}$ is of polynomial size, proving $H \in \mathcal{B}$ is not sufficient to prove termination of the algorithm in polynomial time, especially if some cubes in $\mathcal{B}$ are added and removed several times. Consider for example Fig. 3 which depicts a possible run of the algorithm on three aligned cubes by its successive hypotheses:

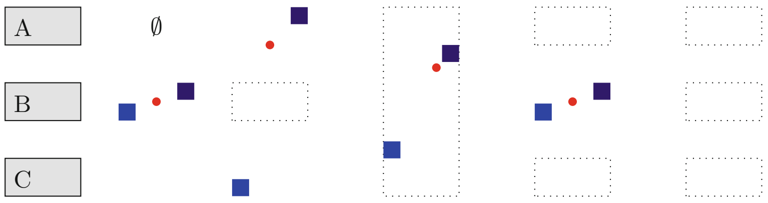


Fig. 3. Possible run on three cubes where cube B is added twice to the hypothesis.

cube B is added during the first step, but is later covered when the algorithm tries to learn A but overshoots. Another overshooting happens when trying to remove the space between A and B, which ends up removing all space between A and C. The cube C has then to be learned a second time, terminating the algorithm.

To circumvent this issue, we propose an optimization that prevents visiting twice the same minimal corner $\underline{\mathbf{v}}$. We base our reasoning on the following observations:

- If $\mathbf{v} \in X$, then $\underline{\mathbf{v}} \in X$, so $\underline{\mathbf{v}}$ should not be later removed.
- If $\mathbf{v} \notin X$, then $\underline{\mathbf{v}} \notin X$, so $\underline{\mathbf{v}}$ should not be later added back to H.

Algorithm 3 introduces an optimized refinement procedure to keep track of the already added maximal corners. Although an analogous optimization can be done on the symmetric difference variant, we only discuss here RefineAddRemove2.

Once a minimal corner $\underline{\mathbf{v}}$ for a candidate cube has been found, we continue the search of a maximal corner $\overline{\mathbf{v}}$ by avoiding points that will result in the removal (resp addition) of already added (resp removed) minimal corners.

Algorithm 3. Optimized refinement avoiding visited minimal corners

```
Let $V \leftarrow \emptyset$
function RefineAddRemove2($H, \mathbf{v}_e, \Phi_X$)
    if $\Phi_X(\mathbf{v}_e)$ then
        Let $\underline{\mathbf{v}} = $ findMinCorner($\mathbf{v}_e, \Phi_{X\setminus H}$)
        Let $\overline{\mathbf{v}} = $ findMaxCorner($\underline{\mathbf{v}}, \Phi_{X\setminus H\setminus\{\mathbf{v} \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}}$)
        $V \leftarrow V \uplus \{\underline{\mathbf{v}}\}$
        return $H \cup \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$
    else
        Let $\underline{\mathbf{v}} = $ findMinCorner($\mathbf{v}_e, \Phi_{H\setminus X}$)
        Let $\overline{\mathbf{v}} = $ findMaxCorner($\underline{\mathbf{v}}, \Phi_{H\setminus X\setminus\{\mathbf{v} \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}}$)
        $V \leftarrow V \uplus \{\underline{\mathbf{v}}\}$
        return $H \setminus \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$
```

Notice how only the maximal corner search benefits from the optimization, by tracking down minimal corners only. As a matter of fact, one could store the whole visited cubes in set V. However, when a search for maximal corner is carried, the resulting cube will intersect a previously visited cube as soon as the max corner crosses the minimal corner of the visited cube.

We exploit again Remark 1 to build an oracle for every mentioned membership oracle. Since V is a finite set, one can indeed build a membership oracle for the set $\{\mathbf{v} \mid \exists \mathbf{v}' \in V \backslash X : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}$. Due to this exclusion region, a finer analysis has to be conducted to prove $H \in \mathcal{B}$.

Lemma 2. *The two optimized variants maintain the following invariants:*

1. $V \cap X \subseteq H$;
2. $(V \backslash X) \cap H = \emptyset$;
3. *for all* $\mathbf{v} \in V$, *and any* k, $\mathbf{v}[k] \in \underline{B}_k$;
4. $H \in \mathcal{B}$.

Properties 1 and 2 ensure that every v added to V is never added twice. These also ensures correctness of the algorithm: remark that the search for a maximal corner is not started from the initial counterexample $\mathbf{v}_e$ but from $\underline{\mathbf{v}}$, which is indeed is in the search space since $\underline{\mathbf{v}} \notin \{\mathbf{v} \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}$ (no point added twice to V). Finally, property 3 ensures that only elements of $(\underline{B}_k)_k$ are added to V, hence a maximal number of $(2n)^d$ additions.

Proof. At the beginning of the algorithm, $V = H = \emptyset$, satisfying all given properties. We prove the result by induction on the iteration step:

1. By definition of corner oracles, namely FINDMAXCORNER, if $\underline{\mathbf{v}} \in X$ has been added to V during some previous iteration, it was added in the first branch (the oracle returns some point in the search region, which excludes X in the second branch). Therefore, it was also added to H during this iteration. Consider some later iteration removing elements from H, namely an iteration executing the second branch. Some cube $C = \text{Cube}(\underline{\mathbf{v}}', \overline{\mathbf{v}}')$ has been computed by the corner oracles in this branch such that $\overline{\mathbf{v}}' \in H \backslash X \backslash \{v \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}}' \leq \mathbf{v}' \leq \mathbf{v}\}$ In particular, since $\underline{\mathbf{v}} \in V$, we do not have $\underline{\mathbf{v}}' \leq \underline{\mathbf{v}} \leq \overline{\mathbf{v}}'$ hence $\underline{\mathbf{v}} \notin C$ and $\underline{\mathbf{v}}$ is not removed.
2. Similar to (1) (symmetric case).
3. For every $\underline{\mathbf{v}}$ added to V, it was produced by a (max) corner query made on $X \backslash H$ or $H \backslash X$. Both of these sets are in $\mathcal{B}$ since $H \in \mathcal{B}$ by induction hypothesis.
4. Let us prove that the cube $C = \text{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$ currently added or removed satisfies $C \in \mathcal{B}$ (hence $H \cup C, H \backslash C \in \mathcal{B}$ which will conclude the induction). We already have proven that $\underline{\mathbf{v}} \in (\underline{B}_k)_k$. We prove now that $\overline{\mathbf{v}} \in (\overline{B}_k)_k$ which is searched over the restricted state space $B = A \backslash \{\mathbf{v} \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}$ for $A = X \backslash H \in \mathcal{B}$ or $A = H \backslash X \mathcal{B}$.
   For any $k \in [1, d]$, $\overline{\mathbf{v}} + \mathbf{e}_k \notin B$ so either:
   - $\overline{\mathbf{v}} + \mathbf{e}_k \notin A \in \mathcal{B}$ so $\overline{\mathbf{v}}[k] \in \overline{B}_k$;
   - or $\overline{\mathbf{v}} + \mathbf{e}_k \in \{\mathbf{v} \mid \exists \mathbf{v}' \in V : \underline{\mathbf{v}} \leq \mathbf{v}' \leq \mathbf{v}\}$ but since $\overline{\mathbf{v}}$ is not in the set, there exists $\mathbf{v}' \in V$ such that $\overline{\mathbf{v}}[k] + 1 = \mathbf{v}'[k]$. Since $\mathbf{v}'[k] \in \underline{B}_k$, we have $\overline{\mathbf{v}}[k] \in \overline{B}_k$.

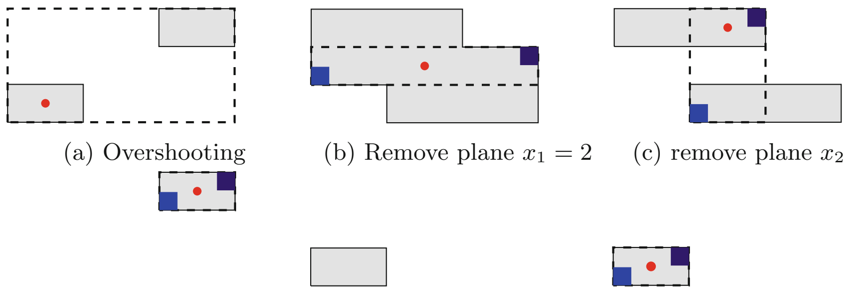
   This concludes the proof.

By combining Proposition 1 and Lemma 2, we summarize the complexity of our overshooting algorithms for a particular target $X = \cup_{i=1}^{n} \text{Cube}(\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i) \in \mathcal{C}_d$.

Theorem 1 (M+EQ). *Both variants of* LearnCubes *terminates in at most* $(2n)^d$ *iterations, where an iteration requires:*

1. *One equivalence query;*
2. *One corner query, or equivalently, a linear number* $O(\text{size}(X))$ *of membership queries.*

This algorithm terminates in polynomial time, for fixed d, in any representation of target X. In particular, the result holds in the worst-case where the representation of X as a finite union of cubes is minimal. As a matter of fact the presented exponential bound in d is tight: there exists a target $X \in \mathcal{C}$ and a pair of corner and equivalence oracles such that both algorithms terminate in exponential time.


(a) Overshooting (b) Remove plane $x_1 = 2$ (c) remove plane $x_2 = 2$

(d) Remove cube (e) Remove cube

Fig. 4. exponential blow-up, case $d = 2$

Example 1. Consider $X = \{\mathbf{0}, \sum_{i=1}^{d} 2\mathbf{e}_i\}$ composed of two cubes, then by learning Cube$(\mathbf{0}, \sum_{i=1}^{d} 2\mathbf{e}_i)$, then removing every middle plane of equation $x_k = 1$ for every $k \in [1, d]$, the resulting hypothesis is composed of $2^d - 2$ cubes to remove. An example with $d = 2$ is depicted in Fig. 4.

Whether finite unions of cubes can be learned in polynomial time in the dimension is left as an open problem, that we relate to DNF formula learning over d variables where each term can be interpreted as a cube over $\{0, 1\}^d$.

4 Extensions

In this section we introduce extensions to the overshooting algorithm from Sect. 3.2. While membership and equivalence queries are sufficient for learning finite sets, one natural extension of the minimal learner setting is to introduce a subset oracle [3]:

Definition 5 (Subset Oracle). *Consider some target concept* $X \in \mathcal{C}_d$ *for some concept class* $\mathcal{C} = \bigcup_{d=1}^{\infty} \mathcal{C}_d$ *and let* $\perp, \top \notin \mathcal{D}$ *be two fresh symbols.*

A subset oracle *(SUB) for* X *is a function* $\rho_X : \mathcal{C}_d \to \{\top, \perp\}$*, which outputs* $\top$ *iff* $H \subseteq X$*.*

The definition is similar to the membership oracle from Definition 1 except the oracle takes a set instead of a single point as input.

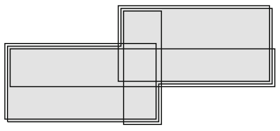
4.1 Maximal Cube Oracle

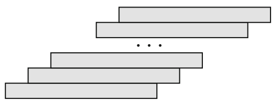
As opposed to the overshooting algorithm, using a subset oracle avoids the overshooting issue, that is to say, we can now search for cubes included in the target X. In order to increase the convergence speed, we nonetheless introduce a maximality criterion on the suitable cubes:

Definition 6 (Maximal Cubes). *A cube* $\mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$ *is* maximal *w.r.t.* X *if*

1. $\mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}}) \subseteq X$
2. For all i, $\mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}} + \mathbf{e}_i) \nsubseteq X$
3. For all i, $\mathrm{Cube}(\underline{\mathbf{v}} - \mathbf{e}_i, \overline{\mathbf{v}}) \nsubseteq X$

Figure 5 provides examples of possible maximal cubes in dimension $d = 2$.


(a) 4 maximal cubes when $n = 2$



(b) $n(n+1)/2$ maximal cubes

Fig. 5. Example of maximal cubes w.r.t. to a union of n cubes

Next, we modify the corner oracle from Sect. 3.1 to use subset queries. Again, we only define the algorithm to find a max corner, the min corner algorithm can be implemented analogously. The algorithm first computes a lower and upper bound for the subsequent binary search. The computation is shown in the function computeMaxBounds. Given a cube defined by its minimal and a maximal corner, the value of coordinate i is increased as long as the resulting cube is still a subset of the target set X. The upper bound $\overline{\mathbf{v}}$ is the first negative reply by the oracle and the lower bound $\underline{\mathbf{v}}$ the last positive response. A binary search is made on $\underline{\mathbf{v}}$ and $\overline{\mathbf{v}}$ in the findMaxIncCorner function.

4.2 Maximal Cube Algorithm

Algorithm 5 presents a procedure that iteratively refines the hypothesis: for any point, the algorithm searches for a maximal cube contained by this point w.r.t. the target and adds it to the hypothesis. One can check that both procedure calls are valid, as $H \subseteq X$ is an invariant. At every iteration the counterexample $\mathbf{v}$ satisfies $\mathbf{v} \in X \setminus H$. The use of the subset oracle ensures that the function FindMaxIncCorner always returns a point $\overline{\mathbf{v}}$ such that $\mathrm{Cube}(\mathbf{v}, \overline{\mathbf{v}}) \subseteq X$. Similarly, the function FindMinIncCorner always returns a corner $\underline{\mathbf{v}}$ such that $\mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}}) \subseteq X$. The resulting cube is then added to the hypothesis, ensuring point $\mathbf{v}$ is never visited again as a counterexample. This entails the termination of the algorithm, in at most $|X|$ iteration of the main loop. A better bound will be explored in Sect. 4.4.

Algorithm 4. Maximal corner of a maximal cube, in $O(\text{size}(X))$ subset queries

```
Ensure: Returned value is a maximal corner of $X$
  function FINDMAXINCCORNER($\underline{\mathbf{v}}, \overline{\mathbf{v}}, \rho_X$)
      for $i \in [1, d]$ do
          $(\underline{b}, \overline{b}) = $ COMPUTEMAXBOUNDS($\underline{\mathbf{v}}, \overline{\mathbf{v}}, i, \rho_X$)
          while $\underline{b} \neq \overline{b}$ do
              $m \leftarrow (\underline{b} + \overline{b}) \div 2$
              if $\rho_X(\text{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}}[i/m]))$ then
                  $\underline{b} \leftarrow m$
              else
                  $\overline{b} \leftarrow m$
          $\overline{\mathbf{v}}[i] \leftarrow \overline{b}$
      return $\overline{\mathbf{v}}$
  function COMPUTEMAXBOUNDS($\underline{\mathbf{v}}, \overline{\mathbf{v}}, i, \rho_X$)
      $\delta \leftarrow 1$
      while $\rho_X(\text{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}} + \delta \cdot \mathbf{e}_i))$ do
          $\delta \leftarrow 2 \cdot \delta$
      return $(\overline{\mathbf{v}}[i] + \delta/2, \overline{\mathbf{v}}[i] + \delta)$
```

Algorithm 5. The maximal cube algorithm

```
  function LEARNMAXCUBE($\rho_X$, $\Psi_X$)
      Let $H \leftarrow \emptyset$
      while $(\mathbf{v} \leftarrow \Psi_X(H)) \neq \top$ do
          Let $\overline{\mathbf{v}} \leftarrow$ FINDMAXINCCORNER($\mathbf{v}, \mathbf{v}, \rho_X$)
          Let $\underline{\mathbf{v}} \leftarrow$ FINDMININCCORNER($\mathbf{v}, \overline{\mathbf{v}}, \rho_X$)
          $H \leftarrow H \cup \text{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$
```

4.3 Extension to the Infinite Case

We discuss now one possible extension to the infinite case, namely when cubes are possibly unbounded and may contain infinitely many points.

We adapt our learning formalism to deal with infinite bounds: for the remainder of the section we extend the discrete lattice $\mathbb{Z}^d$ to $(\mathbb{Z} \uplus \{+\infty, -\infty\})^d$ and extend trivially $\leq$ over the newly introduced points. For $\underline{\mathbf{v}}, \overline{\mathbf{v}} \in (\mathbb{Z} \uplus \{+\infty, -\infty\})^d$, the definition of $C = \text{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$ remains unchanged, in particular $C \subseteq \mathbb{Z}^d$ but may be infinite. The concept class $\mathcal{C}$, hence the domain of oracle functions, is augmented with all finite unions of cubes with (possibly) infinite bounds.

A possible approach to tackle this problem in the minimally adequate teacher (M+EQ) formalism consists in running the overshooting algorithm of Sect. 3 on the state space restricted to some cube of width 2^k centered in $\mathbf{0}$ and gradually increase k if counterexamples outside this restriction are found. This method is discussed in the extended version of the present article [25] but we focus here on a LEARNMAXCUBE adaptation exploiting subset queries (SUB+EQ).

While Algorithm 5 remains unchanged, we need however to adjust the functions FINDMAXINCCORNER and FINDMININCCORNER as those are not able to accelerate the search to infinity. Algorithm 6 achieves this goal by simply overriding the ComputeMaxBounds and ComputeMinBounds subroutines in order to check for possible $+\infty$ and $-\infty$ bounds. Whenever such bound is returned, no further binary search occurs for this coordinate (constant time).

Algorithm 6. Maximal bound overriding, checking for $+\infty$.

```
function computeMaxBounds($\underline{\mathbf{v}}, \overline{\mathbf{v}}, i, \rho_X$)
    if $\rho_X(\mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}}[i/+\infty]))$ then
        return $(+\infty, +\infty)$
    else                    ▷ We refer to original ComputeMaxBounds of Algorithm 4
        return Super.ComputeMaxBounds($\underline{\mathbf{v}}, \overline{\mathbf{v}}, i, \rho_X$)
```

4.4 Complexity

Termination of LearnMaxCube was proved using cardinality arguments in Sect. 4.1. These arguments obviously don't apply in the case where the target set is infinite. Moreover, we are interested in finer complexity analysis.

As in Sect. 3.3, we fix a target representation $X = \cup_{i=1}^{n} \mathrm{Cube}(\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i)$ and study the algorithm complexity with respect to $\sum_{i=1}^{n} \mathrm{size}(\underline{\mathbf{v}}_i) + \mathrm{size}(\overline{\mathbf{v}}_i) \in \mathcal{C}_d$. As some of the vectors $\mathbf{v}$ may contain infinite coordinates, we carefully specify $\mathrm{size}(+\infty) = \mathrm{size}(-\infty) = 1$ and keep the usual definition of $\mathrm{size}(v)$.

Theorem 2 (SUB+EQ). LearnMaxCube *terminates in at most n^{2d} iterations, where an iteration requires:*

1. *One equivalence query;*
2. *One maximal cube query, or equivalently, a linear number $O(\mathrm{size}(X))$ of subset queries.*

Proof. At every iteration, one equivalence query is performed then FindMaxIncCorner and FindMinIncCorner perform a binary search, resulting in a linear number of subset similar (proof similar to Proposition 1).

In order to analyze the number of iterations of the main loop, let us first remark that each added maximal cube is added only once: if we write $\mathbf{v}_k$ the k-th counterexample and C_k the learned maximal cube, then $\mathbf{v}_{k+1} \in X \setminus \cup_{i=1}^{k} C_i$ and $\mathbf{v}_{k+1} \in C_{k+1}$ so $C_{k+1} \neq C_i$ for every $i \in [1, k]$.

The number of iterations is therefore bounded by the number of maximal cubes. We proceed now to bound the number of maximal cubes: Let $C = \mathrm{Cube}(\underline{\mathbf{v}}, \overline{\mathbf{v}})$ be a maximal cube w.r.t. X. For any $k \in [1, d]$ there exist $i, j \in [1, n]$ such that $\underline{\mathbf{v}}[k] = \underline{\mathbf{v}}_i[k]$ and $\overline{\mathbf{v}}[k] = \overline{\mathbf{v}}_j[k]$, hence at most n^2 possibilities for coordinate k.

As in Theorem 1 the number of iterations is polynomial in the number of cubes n but exponential in the dimension d. As opposed to the LearnCubes algorithm, the bound is not tight as the example Fig. 5b provides only a quadratic number of maximal number of cubes. As the maximal cube concept can be related to the notion of *prime implicant*, examples of DNF formula with an exponential of prime implicants (see for example [8]) can be translated into union of cubes with an exponential number of maximal 0–1 cubes.

From a practical perspective, one can nonetheless argue that LearnMaxCube is likely to perform well in practice, by avoiding the overshooting problem mentioned in Example 1 as $H \subseteq X$ is an invariant. In fact, one can easily check that if there are no adjacent[1] cubes, the number of iterations becomes linear.

5 Applications and Experiments

In this section, we describe an immediate application of our learning algorithms to monadic decomposition of quantifier-free Presburger formulas [15,29]. We then report on experimental comparisons between our algorithms and existing methods for the problem.

5.1 Application to Monadic Decomposition

Here we consider quantifier-free linear integer arithmetic formulas without modulo arithmetic:

$$\varphi ::= \alpha_1 \sim \alpha_2 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi,$$

where $\sim \in \{\leq, \geq, =\}$, and α_1, α_2 are integer linear combinations of the variables $x_1, \ldots, x_n$, i.e., α_i is of the form $c_0 + \sum_{j=1}^{n} c_j.x_j$, where each $c_i \in \mathbb{Z}$. The formula $\varphi(\bar{x})$ is said to be *satisfiable* (written $\langle \mathbb{Z}; + \rangle \models \varphi$) if there exists an assignment σ of $\bar{x}$ to $\mathbb{Z}$ such that the formula becomes true. Of course, this is just a simple fragment of the first-order theory of integer linear arithmetic and the notion of $\langle \mathbb{Z}; + \rangle \models \varphi$ can be defined in the same way even with quantifiers [14,18]. A formula φ is said to be *monadic* if it has only one variable. Every monadic formula $\varphi(x)$ in this fragment can be easily transformed into a union integer intervals of the form: (1) $l \leq x \wedge x \leq u$ where $l, u \in \mathbb{Z}$, (2) $l \leq x$ where $l \in \mathbb{Z}$, (3) $x \leq u$ where $u \in Z$, or (4) $\top$ or $\bot$.

A *monadic decomposition* [29] of a formula $\varphi(\bar{x})$ is a boolean combination $\psi(\bar{x})$ of monadic formulas that is equivalent to φ over the theory, i.e., $\langle \mathbb{Z}; + \rangle \models \forall \bar{x}(\varphi \leftrightarrow \psi)$. Of course, not all formulas admit a monadic decomposition (e.g., $x = y$). It was shown in [15] that deciding if a formula in the theory be monadically decomposable is coNP-complete[2]. Veanes *et al.* [29] provides a generic semi-decision procedure for computing a monadic decomposition of a quantifier-free formula as an if-then-else formula that is applicable to pretty much all theories considered in SMT. Despite its genericity, the procedure runs rather well, e.g., as the authors showed on their benchmarking in [29].

[1] Two cubes C_1 and C_2 are *adjacent* if $\min \left\{ \sum_i |\mathbf{v}_1[i] - \mathbf{v}_2[i]| \mid \mathbf{v}_1 \in C_1, \mathbf{v}_2 \in C_2 \right\} \leq 1$.

[2] The proof in [15] uses modulo constraints to show that monadic decomposition of a two-variable formula $\varphi(x, y)$ is coNP-complete. Modulo constraints could be easily removed by allowing more integer variables.

The application of our learning algorithms to computing monadic decomposition arises from the following observation. Since each monadic decomposition can be transformed into DNF, a monadic decomposition of a formula $\varphi(\bar{x})$ over $\langle \mathbb{Z}; + \rangle$ can be constructed as a finite union of (possibly infinite) hypercubes, where an infinite hypercube arises when a variable is either not bounded from above or not bounded from below (or both). Conversely, a finite union H of possibly infinite hypercubes can also be easily transformed into a boolean combination of monadic formulas φ_H. For example, the formula $(0 \leq x \leq 5 \wedge 3 \leq y \leq 10) \vee (8 \leq x)$ corresponds to the union of hypercubes $\text{Cube}((0,3),(5,10)) \cup \text{Cube}((8,-\infty),(+\infty,+\infty))$. Furthermore, all relevant oracles admit a straightforward implementation:

- A membership query $\bar{v}$ requires checking $\langle \mathbb{Z}; + \rangle \models \varphi(\bar{v})$, which can be checked in polynomial-time because φ is quantifier-free.
- An equivalence query H can be reduced to checking
$$\langle \mathbb{Z}; + \rangle \models (\varphi_H \wedge \neg\varphi) \vee (\varphi \wedge \neg\varphi_H).$$
This is a single satisfiability check of quantifier-free integer-linear arithmetic formula, for which highly-optimized solvers exist (e.g., Z3 [26]).
- A subset query H can similarly be reduced to checking
$$\langle \mathbb{Z}; + \rangle \models (\varphi_H \wedge \neg\varphi).$$
This is also a single satisfiability check over $\langle \mathbb{Z}; + \rangle$.

This allows us to apply both of our learning algorithms to the problem.

Monadic decomposition has numerous applications including quantifier elimination [29], string solving [15], and symbolic finite automata/transducers [13,29], among others. In the following example we illustrate how our learning algorithm(s) could be applied to improving quantifier elimination for the theory of linear integer arithmetic.

Example 2. Consider a formula of the form $\forall\bar{x}\exists y\ \varphi(\bar{x},\bar{y})$, where φ is a formula in linear integer arithmetic without modulo constraints. Suppose that φ is monadically decomposable, and is equivalent to the formula $\bigvee_{i=1}^{n} D_i(\bar{x},\bar{y})$, where each D_i is a disjunction of monadic predicates over the variables $\bar{x} \cup \bar{y}$. We assume w.l.o.g. that each D_i is satisfiable. Then, this formula is equisatisfiable (over linear integer arithmetic) to $\psi := \forall\bar{x}\left(\bigvee_{i=1}^{n} D_i(\bar{x},\bar{c}_i)\right)$, where $\bar{y}$ in D_i are replaced by *fresh* constants $\bar{c}_i$ (i.e. two distinct D_i, D_i' use different constants). This can be proven by a simple application of skolemization, and observing that each occurrence of $f(\bar{x})$ in any disjunct is of the form $a < f(\bar{x}) < b$, where $a \in \{-\infty\} \cup \mathbb{Z}$ and $b \in \mathbb{Z} \cup \{\infty\}$, implying that $f(\bar{x})$ can be replaced by a single constant, which does not depend on $\bar{x}$. Finally, let D_i' be the conjuncts in D_i only involving variables in $\bar{x}$. Checking that ψ is true reduces to checking satisfiability of $\bigwedge_{i=1}^{n} \neg D_i'$.

To make this example concrete, we consider the formula $\forall x\exists y(x \geq 0 \rightarrow x + y \geq 5 \wedge y \geq 0)$. A monadic decomposition of the quantifier-free part is $x < 0 \vee \bigvee_{i=0}^{5}(x \geq i \wedge y \geq 5 - i)$. Therefore, checking the above formula can be reduced to satisfiability of $x \geq 0 \wedge \bigwedge_{i=0}^{5} x < i$ which is not satisfiable.

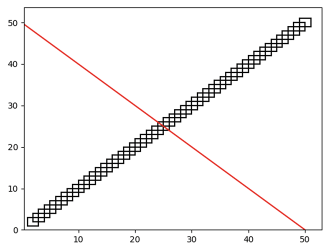
5.2 Experiments

In order to assess the performance of the algorithms FindMaxCorner and FindMinCorner respectively introduced in Sect. 3 and Sect. 4, we consider prototype implementations. The following prototypes and experiments can be found in [24].

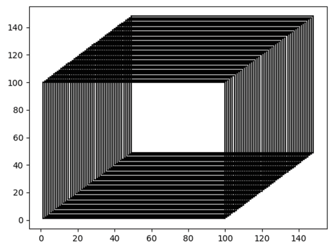
Variants. Although the methods were presented with binary search strategies in mind, we also implemented a more naive unary search procedure to obtain the corners. As later noticed in the experiments, unary search may be preferred for very small cubes and performs especially well for cubes which are based 0–1 integer programs, while binary search achieves better performance for larger cubes. Consequently, we refer to a third variation of the algorithm called "optimized", combining unary search for small instances and binary search for large values. More precisely two variants of the overshooting algorithm from Sect. 3 and three variants of the max cubes algorithm from Sect. 4 are presented, called respectively *overshoot_unary* and *overshoot_binary* and *max_unary*, *max_binary* and *max_optimized*.

Tool Comparison. Evaluation is performed against a generic monadic decomposition procedure *mondec*$_1$ from [29] by Veanes et al., which works over an arbitrary base theory and outputs an if-then-else formula, which could be exponentially more succinct than a formula in DNF. The algorithm, which exploits the python-Z3 framework [26], uses a kind of a decision tree search heuristics to split the input into monadic predicates.

Implementation. Similarly to mondec$_1$, our prototype is implemented in python using the python-Z3 framework, but is specialized in handling linear integer arithmetic formula, and that outputted formulas will be in DNF, unlike mondec$_1$. For monadic decomposition applications, oracles queries are converted to appropriate Z3 satisfaction queries since a (possibly non-monadic) representation of the target set is already known.



(a) 50 overlapping cubes and the diagonal $x + y = 50$.



(b) 100 big Cubes.

Fig. 6. Benchmarks for $\mathbb{Z}^2$.

5.3 Benchmark Suite

Our benchmark suite is restricted to the problem of monadic decomposition of linear integer arithmetic, and its purpose is to stress-test our learning algorithms and mondec$_1$ against various kinds of "extreme conditions". The suite consists of six classes of monadically decomposable example formulas, which were constructed to test five features (see below). Note that the given formulas themselves might contain non-monadic predicates.

The five features (left to right in Table 1) represent the presence of (1) a large amount of cube overlaps, (2) a large number of cubes, (3) a large cube, (4) large dimension, and (5) an unbounded cube. We hypothesized that these five features play important roles in how fast the algorithms perform, which are indeed validated in our experimental results. The six classes of formulas are elaborated below.

Table 1. Features of conducted benchmarks. A "+" (resp. "-") indicates a high (resp. low) presence of a feature.

	Overlap	# Cubes	|Cube|	Dimension	Unbounded
(a)	+	+	−	−	−
(b)	+	−	−	+	−
(c)	+	+	−	−	−
(d)	+	+	+	−	−
(e)	−	+	−	−	−
(f)	+	+	+	−	+

(a) *K Diagonal Restricted* consists of K overlapping cubes of length and width 2 and one diagonal as shown in Fig. 6a. The cubes overlap with at most two other cubes and stack up diagonally. The algorithms need to return all the cubes left of the diagonal.

(b) *10 cubes in* $\mathbb{Z}^d$ consists of $K = 10$ overlapping cubes of size 2^d stacking up diagonally similar to the benchmark K Diagonal Restricted without diagonal restriction.

(c) *K Diagonal Unrestricted* is a variation of Fig. 6a where the algorithms need to return *all* the cubes and all the points on the diagonal.

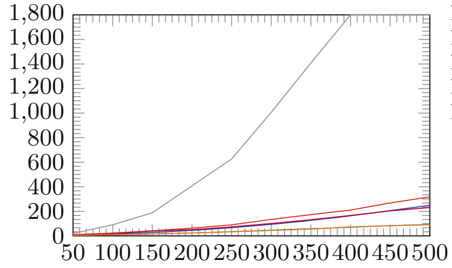
(d) *K Big Overlapping Cube* is a benchmark testing large cubes as depicted in Fig. 6b. It consists of K overlapping cubes of length and width 100 and are overlapping and stacking up diagonally like the benchmark K Diagonal Restricted.

(e) *K Diagonal* is built as the set of points along the diagonal $x = y \leq K$.

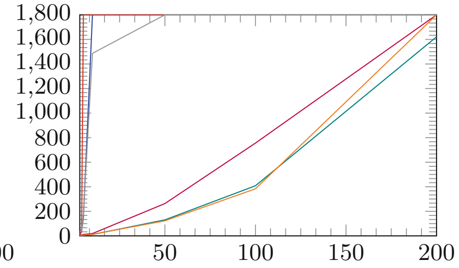
(f) *Example* 2 is generalized to any $K \in \mathbb{N}$ by $x \geq 0 \rightarrow x + y \geq K \wedge y \geq 0$. Its unbounded nature makes it tractable by max_optimized and mondec$_1$ only.

5.4 Results

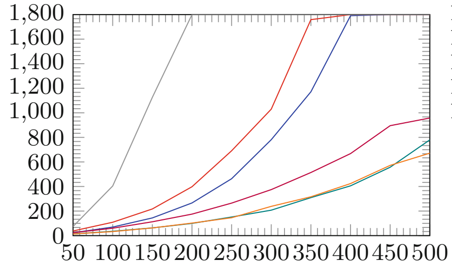
Experiments were conducted on an AMD Ryzen 5 1600 Six-Core CPU with 16 GB of RAM running on Windows 10. The results are summarized in Fig. 7 where each graph represents one benchmark comparing the run times of each algorithm.



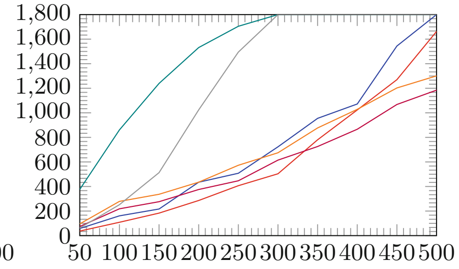
(a) Benchmark on K Diagonal Restricted in $\mathbb{Z}^2$.
The x-axis encodes the amount of cubes K.



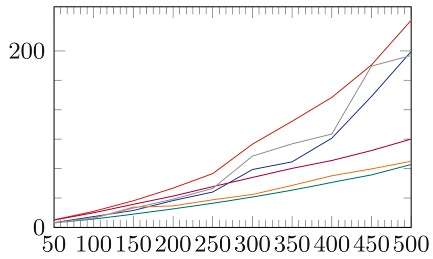
(b) Benchmark on 10 cubes in $\mathbb{Z}^d$.
The x-axis encodes the dimension d.



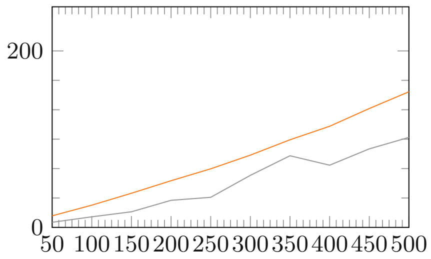
(c) Benchmark on K Diagonal Unrestricted in $\mathbb{Z}^2$.
The x-axis encodes the amount of cubes K.



(d) Benchmark on K Big Cubes in $\mathbb{Z}^2$.
The x-axis encodes the amount of cubes K.



(e) Benchmark on K Diagonal in $\mathbb{Z}^2$.
The x-axis encodes the maximal value for x and y.



(f) Benchmark on Example 2 in $\mathbb{Z}^2$.
The x-axis encodes parameter K.

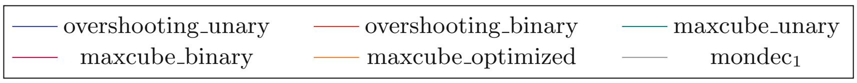


Fig. 7. Benchmark results. The y-axis encodes the time in seconds. The timeout is set to1800 s.

The overshooting phenomenon can be observed in Fig. 7c and Fig. 7e with its quadratic shape, as $d = 2$. In Fig. 7b, the running time quickly diverges as d increases, as anticipated by Example 1.

When the considered cubes are small, as in Fig. 7a and Fig. 7c, the unary search algorithms outperform their binary counterparts, meaning the few additional queries made by the binary search are more costly than a direct enumeration. The optimized variant is therefore a good compromise in all cases.

Figure 7d depicts a benchmark with many large cubes for a fixed dimension. While the impact of the overshooting phenomenon remains contained, the maxcube unary search variant is particularly slow. This can be explained by the size of the cubes making unary search inefficient, combined with the already expensive cost of every single inclusion query.

The mondec_1 algorithm is comparable to the overshooting algorithms in Fig. 7e. It also performs particularly well in Fig. 7f, which we conjecture is due to the conciseness of the solution in if-then-else form used by mondec_1.

Overall, the maxcube algorithm in its optimized form is the most stable algorithm for this benchmark set and should be preferred when an inclusion oracle is available. The extra cost of these queries are here taken into account and remain affordable when implemented with Z3 queries.

6 Conclusion and Future Work

We have presented a polynomial-time algorithm in Angluin's exact learning framework using membership and equivalence for learning a finite union of rectilinear cubes over $\mathbb{Z}^d$ over any fixed dimension d. By considering an additional subset oracle, learning possibly infinite cubes can be achieved with the same complexity, but a simpler and faster learning algorithm in practice. The technique enables the introduction of auxiliary oracles, namely the corner (resp. maximal cube) oracle when a membership (resp. subset) oracle is provided. While oracles for subset queries tend to be difficult to implement, this turns out not to be the case for our proposed application of computing monadic decompositions of quantifier-free integer linear arithmetic formulas without modulo constraints, which is successfully solved by our algorithm.

We mention three future research directions. First, extensions to modulo operations could be explored, by encoding periodicity on d additional coordinates and providing adequate oracles on the encoded target. A second direction consists in applying these learning techniques to the verification of systems by learning invariants which are monadically decomposable in a small number of cubes. Lastly, one promising direction to further improve our algorithms is to investigate how to leverage if-then-else formula representations as used in mondec_1 [29], which could be exponentially more succinct than formulas in DNF.

Acknowledgements. The authors would like to thank the reviewers and program chairs for their thoughtful comments and suggestions to improve the presentation, as well as the Simons Institute for the Theory of Computing and Christoph Haase for fruitful discussions.

References

1. Abe, N.: Characterizing PAC-Learnability of semilinear sets. Inf. Comput. **116**(1), 81–102 (1995)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
3. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
4. Angluin, D., Kharitonov, M.: When won't membership queries help? J. Comput. Syst. Sci. **50**(2), 336–355 (1995)
5. Barceló, P., Hong, C., Le, X.B., Lin, A.W., Niskanen, R.: Monadic decomposability of regular relations. In: 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, 9–12 July 2019, Patras, Greece, pp. 103:1–103:14 (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.103
6. Beyene, T.A., Chaudhuri, S., Popeea, C., Rybalchenko, A.: A constraint-based approach to solving games on infinite graphs. In: Jagannathan, S., Sewell, P. (eds.) The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20–21, 2014, pp. 221–234. ACM (2014). https://doi.org/10.1145/2535838.2535860
7. Carton, O., Choffrut, C., Grigorieff, S.: Decision problems among the main subfamilies of rational relations. ITA **40**(2), 255–275 (2006). https://doi.org/10.1051/ita:2006005
8. Chandra, A.K., Markowsky, G.: On the number of prime implicants. Discrete Math. **24**(1), 7–11 (1978). https://doi.org/10.1016/0012-365X(78)90168-1
9. Chen, Y.-F., Farzan, A., Clarke, E.M., Tsay, Y.-K., Wang, B.-Y.: Learning minimal separating DFA's for compositional verification. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 31–45. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00768-2_3
10. Chen, Y., Hong, C., Lin, A.W., Rümmer, P.: Learning to prove safety over parameterised concurrent systems. In: 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2–6 2017, pp. 76–83 (2017). https://doi.org/10.23919/FMCAD.2017.8102244
11. Chen, Z.: An optimal algorithm for proper learning of unions of two rectangles with queries. In: Du, D.-Z., Li, M. (eds.) COCOON 1995. LNCS, vol. 959, pp. 334–343. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0030848
12. Chen, Z., Ameur, F.: The learnability of unions of two rectangles in the two-dimensional discretized space. J. Comput. Syst. Sci. **59**(1), 70–83 (1999). https://doi.org/10.1006/jcss.1999.1621
13. D'Antoni, L., Veanes, M.: The power of symbolic automata and transducers. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 47–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_3
14. Haase, C.: A survival guide to presburger arithmetic. ACM SIGLOG News **5**(3), 67–82 (2018). https://doi.org/10.1145/3242953.3242964
15. Hague, M., Lin, A.W., Rümmer, P., Wu, Z.: Monadic decomposition in integer linear arithmetic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 122–140. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_8
16. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA, USA (1994)
17. Kopczynski, E., To, A.W.: Parikh images of grammars: complexity and applications. In: 2010 25th Annual IEEE Symposium on Logic in Computer Science, pp. 80–89 (2010). https://doi.org/10.1109/LICS.2010.21

18. Kroening, D., Strichman, O.: Quantified formulas. Decision Procedures. TTC-SAES, pp. 199–227. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-50497-0_9
19. Libkin, L.: Variable independence for first-order definable constraints. ACM Trans. Comput. Log. **4**(4), 431–451 (2003). https://doi.org/10.1145/937555.937557
20. Lin, A.W., Rümmer, P.: Liveness of randomised parameterised systems under arbitrary schedulers. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 112–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_7
21. Löding, C., Madhusudan, P., Neider, D.: Abstract learning frameworks for synthesis. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 167–185. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_10
22. Maass, W., Turán, G.: Algorithms and lower bounds for on-line learning of geometrical concepts. Mach. Learn. **14**(1), 251–269 (1994). https://doi.org/10.1023/A:1022653511837
23. Markgraf, O., Hong, C.-D., Lin, A.W., Najib, M., Neider, D.: Parameterized synthesis with safety properties. In: Oliveira, B.C.S. (ed.) APLAS 2020. LNCS, vol. 12470, pp. 273–292. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64437-6_14
24. Markgraf, O., STAN, D., Lin, A.W.: (Artifact) Learning Union of Integer Hypercubes with Queries (with applications to monadic decomposition) (2021). https://doi.org/10.5281/zenodo.4742954
25. Markgraf, O., Stan, D., Lin, A.W.: Learning union of integer hypercubes with queries (technical report) (2021). https://arxiv.org/abs/2105.13071
26. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
27. Solar-Lezama, A.: Program Synthesis by Sketching. Ph.D. thesis, University of California at Berkele (2008)
28. Takada, Y.: Learning semilinear sets from examples and via queries. Theor. Comput. Sci. **104**(2), 207–233 (1992)
29. Veanes, M., Bjørner, N., Nachmanson, L., Bereg, S.: Monadic decomposition. J. ACM **64**(2), 14:1–14:28 (2017). https://doi.org/10.1145/3040488