A Faster Algorithm for Finding Tarski Fixed Points

John Fearnley
Department of Computer Science, University of Liverpool, UK

Rahul Savani
Department of Computer Science, University of Liverpool, UK

Abstract

Dang et al. have given an algorithm that can find a Tarski fixed point in a k-dimensional lattice of width n using $O(\log^k n)$ queries [2]. Multiple authors have conjectured that this algorithm is optimal [2, 7], and indeed this has been proven for two-dimensional instances [7]. We show that these conjectures are false in dimension three or higher by giving an $O(\log^2 n)$ query algorithm for the three-dimensional Tarski problem, which generalises to give an $O(\log^{k-1} n)$ query algorithm for the k-dimensional problem when $k \geq 3$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases query complexity, Tarski fixed points, total function problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.29

Related Version *Full Version*: https://arxiv.org/abs/2010.02618

Acknowledgements We would like to thank Kousha Etessami, Thomas Webster, and an anonymous reviewer for pointing out that the proof of Lemma 12 could be drastically simplified from its original version.

1 Introduction

Tarski's fixed point theorem states that every order preserving function on a complete lattice has a greatest and least fixed point [11], and therefore in particular, every such function has at least one fixed point. Recently, there has been interest in the complexity of finding such a fixed point. This is due to its applications, including computing Nash equilibria of supermodular games and finding the solution of a simple stochastic game [7].

Prior work has focused on the complete lattice L defined by a k-dimensional grid of width n. Dang, Qi, and Ye [2] give an algorithm that finds a fixed point of a function $f : L \to L$ using $O(\log^k n)$ queries to f. This algorithm uses recursive binary search, where a k-dimensional problem is solved by making $\log n$ recursive calls on $(k-1)$-dimensional sub-instances. They conjectured that this algorithm is optimal.

Later work of Etessami, Papadimitriou, Rubinstein, and Yannakakis took the first step towards proving this [7]. They showed that finding a Tarski fixed point in a two-dimensional lattice requires $\Omega(\log^2 n)$ queries, meaning that the Dang et al. algorithm is indeed optimal in the two-dimensional case. Etessami et al. conjectured that the Dang et al. algorithm is optimal for constant k, and they leave as an explicit open problem the question of whether their lower bound can be extended to dimension three or beyond.

Our contribution. In this paper we show that, surprisingly, the Dang et al. algorithm is not optimal in dimension three, or any higher dimension, and so we falsify the prior conjectures. We do this by giving an algorithm that can find a Tarski fixed point in three dimensions using $O(\log^2 n)$ queries, thereby beating the $O(\log^3 n)$ query algorithm of Dang et al. Our new algorithm can be used as a new base case for the Dang et al. algorithm, and this leads to a $O(\log^{k-1} n)$ query algorithm for k-dimensional instances when $k \geq 3$, which saves a $\log n$ factor over the $O(\log^k n)$ queries used by the Dang et al. algorithm.

The Dang et al. algorithm solves a three-dimensional instance by making recursive calls to find a fixed point of $\log n$ distinct two-dimensional sub-instances. Our key innovation is to point out that one does not need to find a fixed point of the two-dimensional sub-instance to make progress. Instead, we define the concept of an *inner algorithm* (Definition 3) that, given a two-dimensional sub-instance, is permitted to return any point that lies in the up or down set of the three-dimensional instance (defined formally later). This is a much larger set of points, so whereas finding a fixed point of a two-dimensional instance requires $\Omega(\log^2 n)$ queries [7], we give a $O(\log n)$ query inner algorithm for two-dimensional instances. This inner algorithm is quite involved, and is the main technical contribution of the paper.

We show that, given an inner algorithm for dimension $k-1$, a reasonably straightforward *outer algorithm* can find a Tarski fixed point by making $O(k \cdot \log n)$ calls to the inner algorithm. Thus we obtain a $O(\log^2 n)$ query algorithm for the case where $k = 3$. We leave as an open problem the question of whether efficient inner algorithms exist in higher dimensions.

Though we state our results in terms of query complexity for the sake of simplicity, it should be pointed out that both our outer and inner algorithms run in polynomial time. Specifically, our algorithms will run in $O(\text{poly}(\log n, k) \cdot \log^{k-1} n)$ time when the function is presented as a Boolean circuit of size $\text{poly}(\log n, k)$.

Related work. Etessami et al. also studied the computational complexity of the Tarski problem [7], showing that the problem lies in PPAD and PLS. However, the exact complexity of the problem remains open. It is not clear whether the problem is PPAD ∩ PLS-complete [8], or contained in some other lower class such as EOPL or UEOPL [9].

Tarski's fixed point theorem has been applied in a wide range of settings within Economics [12, 10, 13], and in particular to settings that can be captured by supermodular games, which are in fact equivalent to the Tarski problem [7]. In terms of algorithms, Echenique [6] studied the problem of computing all pure equilibria of a supermodular game, which is at least as hard as finding the greatest or least fixed point of the Tarski problem, which is itself NP-hard [7]. There have also been several papers that study properties of Tarski fixed points, such as the complexity of deciding whether a fixed point is unique [2, 4, 3, 5]. The Tarski problem has also been studied in the setting where the partial order is given by an oracle [1].

2 Preliminaries

Lattices. We work with a complete lattice defined over a k-dimensional grid of points. We define $\text{Lat}(n_1, n_2, \ldots, n_k)$ to be the k-dimensional lattice with side-lengths given by $n_1, \ldots, n_k$. That is, $\text{Lat}(n_1, n_2, \ldots, n_k)$ contains every $x \in \mathbb{N}^k$ such that $1 \leq x_i \leq n_i$ for all $i = 1, \ldots, k$. Throughout, we use k to denote the dimensionality of the lattice, and $n = \max_{i=1}^{k} n_i$ to be the width of the widest dimension. We use $\preceq$ to denote the natural partial order over this lattice with $x \preceq y$ if and only if $x, y \in L$ and $x_i \leq y_i$ for all $i \leq k$.

The Tarski fixed point problem. Given a lattice L, a function $f : L \to L$ is *order preserving* if $f(x) \preceq f(y)$ whenever $x \preceq y$. A point $x \in L$ is *fixed point* of f if $f(x) = x$. A weak version of Tarski's theorem can be stated as follows.

▶ **Theorem 1** ([11]). *Every order preserving function on a complete lattice has a fixed point.*

Thus, we can define a total search problem for Tarski's fixed point theorem.

▶ **Definition 2** (Tarski). *Given a lattice L, and a function $f : L \to L$, find one of:*
(T1) *A point $x \in L$ such that $f(x) = x$.*
(T2) *Two points $x, y \in L$ such that $x \preceq y$ and $f(x) \not\preceq f(y)$.*

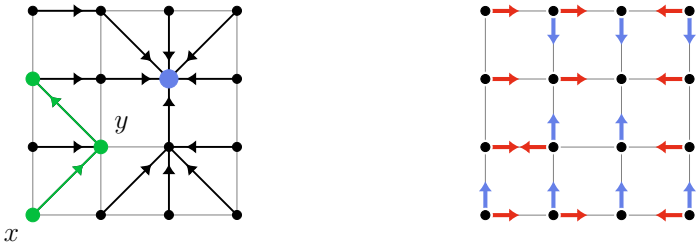

Figure 1 Left: a TARSKI instance. Right: our diagramming notation for the same instance.

Solutions of type 1 are fixed points of f, whereas solutions of type 2 witness that f is not an order preserving function. By Tarski's theorem, if a function f has no solutions of type 2, then it must have a solution of type 1, and so TARSKI is a total problem.

The left-hand picture in Figure 1 gives an example of a two-dimensional TARSKI instance. The blue point is a fixed point, and so is a 1 solution, while the highlighted green arrows give an example of an order preservation violation, and so (x, y) is a 2 solution.

Throughout the paper we will use a diagramming notation, shown on the right in Figure 1, that decomposes the dimensions of the instance. The red arrows correspond to dimension 1, where an arrow pointing to the left indicates that $f(x)_1 \leq x_1$, while an arrow to the right[1] indicates that $x_1 \leq f(x)_1$. Blue arrows do the same thing for dimension 2, and we will use green arrows for dimension 3 in the cases where this is relevant.

The up and down sets. Given a function f over a lattice L, we define $\text{Up}(f) = \{x \in L : x \preceq f(x)\}$, and $\text{Down}(f) = \{x \in L : f(x) \preceq x\}$. We call $\text{Up}(f)$, the *up set*, which contains all points in which f goes up according to the ordering $\preceq$, and likewise we call $\text{Down}(f)$ the *down set*. Note that the set of fixed points of f is exactly $\text{Up}(f) \cap \text{Down}(f)$.

Slices. A *slice* of the lattice L is defined by a tuple $s = (s_1, s_2, \ldots, s_k)$, where each $s_i \in \mathbb{N} \cup \{*\}$. The idea is that, if $s_i \neq *$, then we fix dimension i of L to be s_i, and if $s_i = *$, then we allow dimension i of L to be free. Formally, we define the sliced lattice $L_s = \{x \in L : x_i = s_i \text{ whenever } s_i \neq *\}$. We say that a slice is a *principle slice* if it fixes exactly one dimension and leaves the others free. For example $(1, *, *)$, $(*, 33, *)$, and $(*, *, 261)$ are all principle slices of a three-dimensional lattice.

Given a slice s, and a function $f : L \to L$, we define $f_s : L_s \to L_s$ to be the *restriction* of f to L_s. Specifically, for each $x \in L_s$, we define $(f_s(x))_i = f(x)_i$ if $s_i = *$, and $(f_s(x))_i = s_i$ otherwise. This definition projects the function f down onto the slice s.

A fact that we will use repeatedly in the paper is that an order preservation violation in a slice s is also an order preservation violation for the whole instance. More formally, if $x, y \in L_s$ satisfy $x \preceq y$ and $f_s(x) \not\preceq f_s(y)$, then we also have $f(x) \not\preceq f(y)$, since there exists a dimension i such that $f(x)_i = f_s(x)_i > f_s(y)_i = f(y)_i$.

Sub-instances. A *sub-instance* of a lattice L is defined by two points $x, y \in L$ that satisfy $x \preceq y$. Informally, the sub-instance defined by x and y is the lattice containing all points between x and y. Formally, we define $L_{x,y} = \{a \in L : x \preceq a \preceq y\}$.

[1] If $x_1 = f(x)_1$ we could use either arrow, but will clarify in the text whenever this ambiguity matters.

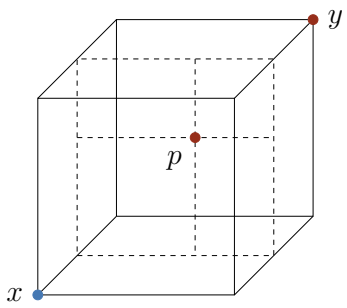

Figure 2 One iteration of the outer algorithm. The dashed lines show the principle slice chosen by the algorithm, and the point p is the point returned by the inner algorithm. In this case $p \in \text{Down}(f)$, and so the algorithm focuses on the sub-instance $L_{x,p}$.

3 The Outer Algorithm

The task of the outer algorithm is to find a solution to the TARSKI instance by making $O(k \cdot \log n)$ calls to the inner algorithm. We state our results for dimension k, even though we only apply the outer algorithm with $k = 3$, since, in the future, an efficient inner algorithm in higher dimensions may be found. Formally, an inner algorithm is defined as follows.

▶ **Definition 3** (Inner algorithm). *An inner algorithm for a TARSKI instance takes as input a sub-instance $L_{a,b}$ with $a \in \text{Up}(f)$ and $b \in \text{Down}(f)$, and a principle slice s of that sub-instance. It outputs one of the following.*

- *A point $x \in L_{a,b} \cap L_s$ such that $x \in \text{Up}(f)$ or $x \in \text{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

It is important to understand that here we are looking for points that lie in the up or down set of the *three-dimensional* instance, a point that lies in $\text{Up}(f_s)$ but for which f goes down in the third dimension would not satisfy this criterion.

The algorithm. Throughout the outer algorithm, we will maintain two points $x, y \in L$ with the invariant that $x \preceq y$ and $x \in \text{Up}(f)$ and $y \in \text{Down}(f)$. The following lemma implies that if x and y satisfy the invariant, then $L_{x,y}$ must contain a solution to the TARSKI problem. This will allow us to focus on smaller and smaller instances that are guaranteed to contain a solution.

▶ **Lemma 4.** *Let L be a lattice and $f : L \to L$ be a TARSKI instance. If there are two points $a, b \in L$ satisfying $a \preceq b$, $a \in \text{Up}(f)$, and $b \in \text{Down}(f)$, then one of the following exists.*

- *A point $x \in L_{a,b}$ satisfying $f(x) = x$.*
- *Two points $x, y \in L_{a,b}$ satisfying $x \preceq y$ and $f(x) \not\preceq f(y)$.*

Moreover, there is an algorithm that finds one of the above using $O(\sum_{i=1}^{k}(a_i - b_i))$ queries.

Initially we set $x = (1, 1, \ldots, 1)$, which is the least element, and $y = (n_1, n_2, \ldots, n_k)$, which is the greatest element. Note that $x \preceq f(x)$ holds because x is the least element, and likewise $f(y) \preceq y$ holds because y is the greatest element, so the invariant holds for these two points.

Each iteration of the outer algorithm will reduce the number of points in $L_{x,y}$ by a factor of two. To do this, the algorithm selects a largest dimension of that sub-instance, which is a dimension i that maximizes $y_i - x_i$. It then makes a call to the inner algorithm for the principle slice s defined so that $s_i = \lfloor (y_i - x_i)/2 \rfloor$ and $s_j = *$ for all $j \neq i$.

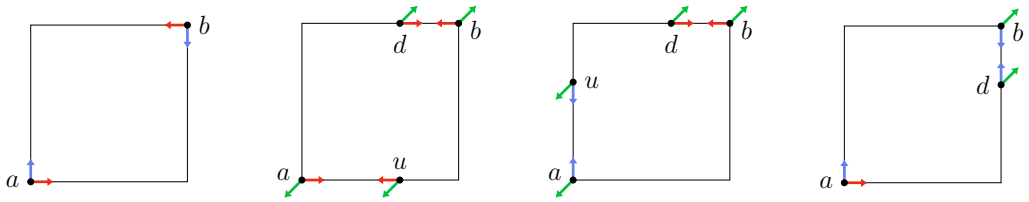

Figure 3 Four example sub-instances that satisfy the inner algorithm invariant.

1. If the inner algorithm returns a violation of order preservation in the slice, then this is also an order preservation violation in L, and so the algorithm returns this and terminates.
2. If the inner algorithm returns a point p in the slice such that $p \in \text{Up}(f)$, then the algorithm sets $x := p$ and moves to the next iteration.
3. If the inner algorithm returns a point p such that $p \in \text{Down}(f)$, then the algorithm sets $y := p$ and moves to the next iteration.

Figure 2 gives an example of this procedure.

The algorithm can continue as long as there exists a dimension i such that $y_i - x_i \geq 2$, since this ensures that there will exist a principle slice strictly between x and y in dimension i that cuts the sub-instance in half. Note that there can be at most $k \cdot \log n$ iterations of the algorithm before we arrive at the final sub-instance $L_{x,y}$ with $y_i - x_i < 2$ for all i. Lemma 4 gives us an efficient algorithm to find a solution in this final instance, which uses at most $O(\sum_{i=1}^{k}(y_i - x_i)) = O(k)$ queries. So we have proved the following theorem.

▶ **Theorem 5.** *Suppose that there exists an inner algorithm that makes at most q queries. Then a solution to the* Tarski *problem can be found by making $O(q \cdot k \cdot \log n + k)$ queries.*

4 The Inner Algorithm

We now describe an inner algorithm for three dimensions that makes $O(\log n)$ queries. Throughout this section we assume that the inner algorithm has been invoked on a sub-instance $L_{u,d}$ and principle slice s, and without loss of generality we assume that $s = (*, *, s_3)$.

Down set witnesses. Like the outer algorithm, the inner algorithm will also focus on smaller and smaller sub-instances that are guaranteed to contain a solution by an invariant, but now the invariant is more complex. To define the invariant, we first introduce the concept of a *down set witness* and an *up set witness*. The points d and b in the second example in Figure 3 give an example of a down set witness. Note that the following properties are satisfied.

- f weakly increases at d and b in dimension 3.
- d and b have the same coordinate in dimension 2.
- d weakly increases in dimension 1 while b weakly decreases in dimension 1.

We also allow down set witnesses like those given by d and b in the fourth example of Figure 3 that satisfy the same properties with dimensions 1 and 2 swapped. Thus, the formal definition of a down set witness abstracts over dimensions 1 and 2.

▶ **Definition 6** (Down set witness). *A down set witness is a pair of points (d, b) with $d, b \in L_s$ such that both of the following are satisfied.*

- $d_3 \leq f(d)_3$ *and* $b_3 \leq f(b)_3$.
- $\exists\ i, j \in \{1, 2\}$ *with* $i \neq j$ *s.t.* $d_i = b_i$ *and* $d_j \leq b_j$, *while* $d_j \leq f(d)_j$ *and* $f(b)_j \leq b_j$.

If (d, b) is a down set witness and $d_2 = b_2$, then we call (d, b) a *top-boundary* witness, while if $d_1 = b_1$, then we call (d, b) a *right-boundary* witness.

The following lemma states that if we have a down set witness (d, b), then between d and b we can find either a solution that can be returned by the inner algorithm (cases 1 and 2 of the lemma), or a point that is in the down set of the slice s (case 3 of the lemma).

Informally, the proof for a top-boundary witness (d, b) uses the fact that d and b point towards each other in dimension 1 to argue that there must be a fixed point p (or an order preservation violation) of the one-dimensional slice between d and b. Then, it is shown that either this point is in $\mathrm{Up}(f)$, and so is a solution for the inner algorithm, or it is in $\mathrm{Down}(f_s)$, or that p violates order preservation with d or b.

▶ **Lemma 7.** *If (d, b) is a down set witness, then one of the following exists.*
1. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Up}(f)$.*
2. *Two points x, y satisfying $d \preceq x \preceq y \preceq b$ that witness order preservation violation of f.*
3. *A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Down}(f_s)$.*

Up set witnesses. An up set witness is simply a down set witness in which all inequalities have been flipped. The second and third diagrams in Figure 3 show the two possible configurations of an up set witness (a, u). Note that for up set witnesses, dimension 3 is now required to weakly decrease.

▶ **Definition 8** (Up set witness). *An up set witness is a pair of points (a, u) with $a, u \in L_s$ such that both of the following are satisfied.*
- *$a_3 \geq f(a)_3$ and $u_3 \geq f(u)_3$.*
- *$\exists\ i, j \in \{1, 2\}$ with $i \neq j$ s.t. $a_i = u_i$ and $u_j \geq a_j$, while $u_j \geq f(u)_j$ and $f(a)_j \geq a_j$.*

We say that an up set witness (a, b) is a *left-boundary* witness if $a_1 = b_1$, while we call it a *bottom-boundary* witness if $a_2 = b_2$.

The following lemma is the analogue of Lemma 7 for up set witnesses. The proof simply flips all inequalities in the proof of Lemma 7.

▶ **Lemma 9.** *If (a, u) is an up set witness, then one of the following exists.*
1. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \mathrm{Down}(f)$.*
2. *Two points x, y satisfying $a \preceq x \preceq y \preceq u$ that witness order preservation violation of f.*
3. *A point c satisfying $a \preceq c \preceq u$ such that $c \in \mathrm{Up}(f_s)$.*

The invariant. At each step of the inner algorithm, we will have a sub-instance $L_{a,b}$ that satisfies the following invariant.

▶ **Definition 10** (Inner algorithm invariant). *The instance $L_{a,b}$ satisfies the invariant if*
- *Either $a \in \mathrm{Up}(f_s)$ or there is a known up set witness (a, u) with $u \preceq b$.*
- *Either $b \in \mathrm{Down}(f_s)$ or there is a known down set witness (d, b) with $a \preceq d$.*

If we have both an up set witness and a down set witness then we also require that $u \preceq d$.

Figure 3 gives four example instances that satisfy the invariant. Note that there are actually nine possible configurations, since the first point of the invariant can be satisfied either by a point in the up set, a left-boundary up set witness, or a bottom-boundary up set witness, and the second point of the invariant likewise has three possible configurations.

The following lemma shows that, if the invariant is satisfied, then the sub-instance $L_{a,b}$ contains a solution that can be returned by the inner algorithm. The proof invokes Lemmas 7 and 9 to either immediately find a solution for the inner algorithm, or find two points $x \preceq y$ in the sub instance where x is in the up set and y is in the down set. The latter case allows us to invoke Lemma 4 to argue that the sub-instance contains a fixed point p of the slice s. If p weakly increases in the third dimension, then $p \in \mathrm{Up}(f)$, while if p decreases in the third dimension then $p \in \mathrm{Down}(f)$.

▶ **Lemma 11.** *If $L_{a,b}$ satisfies the invariant then one of the following exists.*

- *A point $x \in L_{a,b}$ such that $x \in \text{Up}(f)$ or $x \in \text{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

A special case. There is a special case that we will encounter in the inner algorithm that requires more effort to deal with. One example of this case is shown in Case 3.a.ii of Figure 6. Here we have a point p on the right-hand boundary of the instance that satisfies $p_1 < f(p)_1$, meaning that f moves p outside of the instance. If $b \in \text{Down}(f)$, or if there is a top-boundary down set witness, then it is straightforward to show that p and b violate order preservation.

However, if we have a right-boundary down set witness (d, b) with $p \preceq d$ then we need to do further work[2]. Note that the properties of a down set witness ensure that $d_2 \leq f(d)_2$ and $d_3 \leq f(d)_3$. But there are two possibilities for dimension 1. If $d_1 \leq f(d)_1$ then $d \in \text{Up}(f)$ and it can be returned by the inner algorithm. On the other hand, if $d_1 > f(d)_1$, then we can show that p and d violate order preservation. We prove this formally in the following lemma.

▶ **Lemma 12.** *Let $L_{a,b}$ be a sub-instance that satisfies the invariant, and let p be a point satisfying $a \preceq p \preceq b$ that also satisfies one of the following conditions.*

1. *$p_1 = b_1$ and $p_1 < f(p)_1$.*
2. *$p_2 = b_2$ and $p_2 < f(p)_2$.*
3. *$p_1 = a_1$ and $p_1 > f(p)_1$.*
4. *$p_2 = a_2$ and $p_2 > f(p)_2$.*

Suppose further that, if there exists a down-set witness (d, b) then $p \preceq d$, and if there exists an up-set witness (a, u) then $u \preceq p$. Then there exists a solution for the inner algorithm that can be found using constantly many queries.

Initialization. The input to the algorithm is a sub-instance $L_{x,y}$, and recall that we have fixed the principle slice $s = (*, *, s_3)$. The initial values for a and b are determined as follows. For each dimension i we have $a_i = s_3$ if $i = 3$, and $a_i = x_i$ otherwise, and we have $b_i = s_3$ if $i = 3$, and $b_i = y_i$ otherwise. That is, a and b are the projections of x and y onto s.

The following lemma states that either a and b satisfy the invariant, or that we can easily find a violation of order preservation.

▶ **Lemma 13.** *Either $L_{a,b}$ satisfies the invariant, or there is violation of order preservation between a and x, or between b and y.*

The algorithm. Now suppose that we have an instance $L_{a,b}$ that satisfies the invariant. We will describe how to execute one iteration of the algorithm, which will either find a violation of order preservation, or find a new instance whose size is at most half the size of the $L_{a,b}$.

We begin by defining some important points. We define $\mathsf{mid} = \lfloor (a + b)/2 \rfloor$ to be the *midpoint* of the instance, and we define the following points, which are shown in Figure 4:

$$\mathsf{bot} = (\lfloor (a_1 + b_1)/2 \rfloor, a_2), \qquad \mathsf{left} = (a_1, \lfloor (a_2 + b_2)/2 \rfloor),$$
$$\mathsf{top} = (\lfloor (a_1 + b_1)/2 \rfloor, b_2), \qquad \mathsf{right} = (b_1, \lfloor (a_2 + b_2)/2 \rfloor).$$

[2] The case where $p \succ d$ will never occur in our algorithm, so we can ignore it.

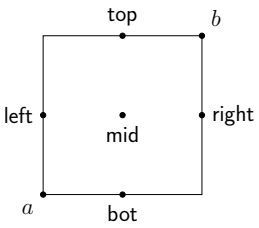

Figure 4 The five points used by the inner algorithm.

Step 1: Fixing the up and down set witnesses. Suppose that $L_{a,b}$ satisfies the invariant with a top-boundary down set witness (d, b), We would like to ensure that $\mathsf{top} \preceq d$, since otherwise if we cut the instance in half in a later step, the witness may no longer be within the sub-instance. For the same reason, we would like to ensure that (d, b) satisfies $\mathsf{right} \preceq d$ for a right-boundary down set witness, that (a, u) satisfies $u \preceq \mathsf{left}$ for a left-boundary up set witness, and that (a, u) satisfies $u \preceq \mathsf{bot}$ for a bottom-boundary up set witness. By the end of Step 1 we will have either found a violation of order preservation, moved into the next iteration with a sub-instance of half the size, or all inequalities above will hold.

Step 1 consists of the following procedure. The procedure should be read alongside Figure 5, which gives a diagram for every case presented below.

1. If (d, b) is a top-boundary down set witness and $\mathsf{top} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \mathsf{top}$ we use the following procedure.
   a. We first check if $\mathsf{top}_3 > f(\mathsf{top})_3$. If this is the case, then since the invariant ensures that $d_3 \leq f(d)_3$ we have $f(d)_3 \geq d_3 = \mathsf{top}_3 > f(\mathsf{top})_3$ so $d \preceq \mathsf{top}$ but $f(d) \not\preceq f(\mathsf{top})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\mathsf{top}_1 > f(\mathsf{top})_1$. In this case, we can use (d, top) as a down set witness for the sub-instance $L_{a,\mathsf{top}}$, where we observe that top satisfies the requirements since $\mathsf{top}_3 \leq f(\mathsf{top})_3$ and $\mathsf{top}_1 > f(\mathsf{top})_1$. Hence, $L_{a,\mathsf{top}}$ satisfies the invariant (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{a,\mathsf{top}}$.
   c. In this final case we have $\mathsf{top}_3 \leq f(\mathsf{top})_3$ and $\mathsf{top}_1 \leq f(\mathsf{top})_1$. Therefore (top, b) is a valid down set witness for $L_{a,b}$ (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d \prec \mathsf{top}$). So we can replace (d, b) with (top, b) and continue, noting that our down set witness now satisfies the required inequality.
2. If (d, b) is a right-boundary down set witness and $\mathsf{right} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \mathsf{right}$ then we use the same procedure as case 1, where dimensions 1 and 2 are exchanged and the point top is replaced by the point right.
3. If (a, u) is a bottom-boundary up set witness and $u \preceq \mathsf{bot}$ then there is no need to do anything. On the other hand, if $\mathsf{bot} \prec u$ then we use the following procedure, which is the same as the procedure from case 1, where all inequalities have been flipped.
   a. We first check if $\mathsf{bot}_3 < f(\mathsf{bot})_3$. If this is the case, then since the invariant ensures that $u_3 \geq f(u)_3$ we have $f(u)_3 \leq u_3 = \mathsf{bot}_3 < f(\mathsf{bot})_3$ so $u \succeq \mathsf{bot}$ but $f(u) \not\succeq f(\mathsf{bot})$, and an order preservation violation has been found and the inner algorithm terminates.
   b. We next check the whether $\mathsf{bot}_1 < f(\mathsf{bot})_1$. In this case, we can use (bot, u) as an up set witness for the sub-instance $L_{\mathsf{bot},b}$, where we observe that bot satisfies the requirements since $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$ and $\mathsf{bot}_1 < f(\mathsf{bot})_1$. Hence, $L_{\mathsf{bot},b}$ satisfies the invariant (if $L_{a,b}$ also has a down set witness (d, b) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

   c. In this final case we have $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$ and $\mathsf{bot}_1 \geq f(\mathsf{bot})_1$. Therefore (a, bot) is a valid up set witness for $L_{a,b}$ (if $L_{a,b}$ also has a down set witness (d, b) then we note that $\mathsf{bot} \preceq u \preceq d$). So we can replace (a, u) with (a, bot), noting that our up set witness now satisfies the required inequality.

4. If (a, u) is a left-boundary up set witness and $u \preceq \mathsf{left}$ then there is no need to do anything. On the other hand, if $u \succ \mathsf{left}$ then we use the same procedure as case 3, where dimensions 1 and 2 are exchanged and the point left is replaced by the point bot.

Step 2: Find a smaller sub-instance. If Step 1 of the algorithm did not already move us into the next iteration of the algorithm with a smaller instance, we apply Step 2. This step performs a case analysis on the point mid. The following procedure should be read in conjunction with Figure 6, which provides a diagram for every case.

1. Check if $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \leq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Up}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{\mathsf{mid},b}$. Note that if $L_{a,b}$ has a down-set witness (d, b), then Step 1 of the algorithm has ensured that $\mathsf{mid} \preceq d$, and so (d, b) is also a valid down-set witness for $L_{\mathsf{mid},b}$.
2. Check if $\mathsf{mid}_1 \geq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Down}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{a,\mathsf{mid}}$. Note that if $L_{a,b}$ has an up-set witness (a, u), then Step 1 of the algorithm has ensured that $u \preceq \mathsf{mid}$, and so (a, u) is also a valid down-set witness for $L_{a,\mathsf{mid}}$.
3. Check if $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 > f(\mathsf{mid})_2$. If so, we use the following procedure.
   a. Check if $\mathsf{mid}_3 \leq f(\mathsf{mid})_3$. If so, do the following.
      i. Check if $\mathsf{right}_3 > f(\mathsf{right})_3$. If this holds then we have $f(\mathsf{mid})_3 \geq \mathsf{mid}_3 = \mathsf{right}_3 > f(\mathsf{right})_3$, meaning that $\mathsf{mid} \preceq \mathsf{right}$ but $f(\mathsf{mid}) \not\preceq f(\mathsf{right})$. Thus we have found a violation of order preservation and the algorithm terminates.
      ii. Check if $\mathsf{right}_1 < f(\mathsf{right})_1$. If this holds then we use Lemma 12 (with $p := \mathsf{right}$) to find a solution that can be returned by the inner algorithm.
      iii. If we reach this case then we have $\mathsf{mid}_3 \leq f(\mathsf{mid})_3$ and $\mathsf{right}_3 \leq f(\mathsf{right})_3$, while we also have $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$ and $\mathsf{right}_1 \geq f(\mathsf{right})_1$. Thus $(\mathsf{mid}, \mathsf{right})$ is a valid down set witness for the instance $L_{a,\mathsf{right}}$. Note that if $L_{a,b}$ has an up set witness (a, u), then Step 1 ensures that $u \preceq \mathsf{mid}$, and so $L_{a,\mathsf{right}}$ satisfies the invariant. So the algorithm moves to the next iteration with sub-instance $L_{a,\mathsf{right}}$.
   b. In this case we have $\mathsf{mid}_3 > f(\mathsf{mid})_3$. The following three steps are symmetric to those used in Case 3.a, but with all inequalities flipped, dimension 1 substituted for dimension 2, the point bot substituted for right, and the point a substituted for b.
      i. Check if $\mathsf{bot}_3 > f(\mathsf{bot})_3$. If this holds then we have $f(\mathsf{mid})_3 \leq \mathsf{mid}_3 = \mathsf{bot}_3 < f(\mathsf{bot})_3$, meaning that $\mathsf{mid} \succeq \mathsf{bot}$ but $f(\mathsf{mid}) \not\succeq f(\mathsf{bot})$. Thus we have found a violation of order preservation and the algorithm terminates.
      ii. Check if $\mathsf{bot}_2 > f(\mathsf{bot})_2$. If this holds then we can use Lemma 12 (with $p := \mathsf{bot}$) to find a solution that can be returned by the inner algorithm.
      iii. If we reach this case then we have $\mathsf{mid}_3 \geq f(\mathsf{mid})_3$ and $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$, while we also have $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$ and $\mathsf{bot}_2 \leq f(\mathsf{bot})_2$. Thus $(\mathsf{bot}, \mathsf{mid})$ is a valid up set witness for the instance $L_{\mathsf{bot},b}$. Note that if $L_{a,b}$ has a down set witness (d, b), then Step 1 of the algorithm ensures that $d \succeq \mathsf{mid}$, and so $L_{\mathsf{bot},b}$ satisfies the invariant. The algorithm will therefore move to the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

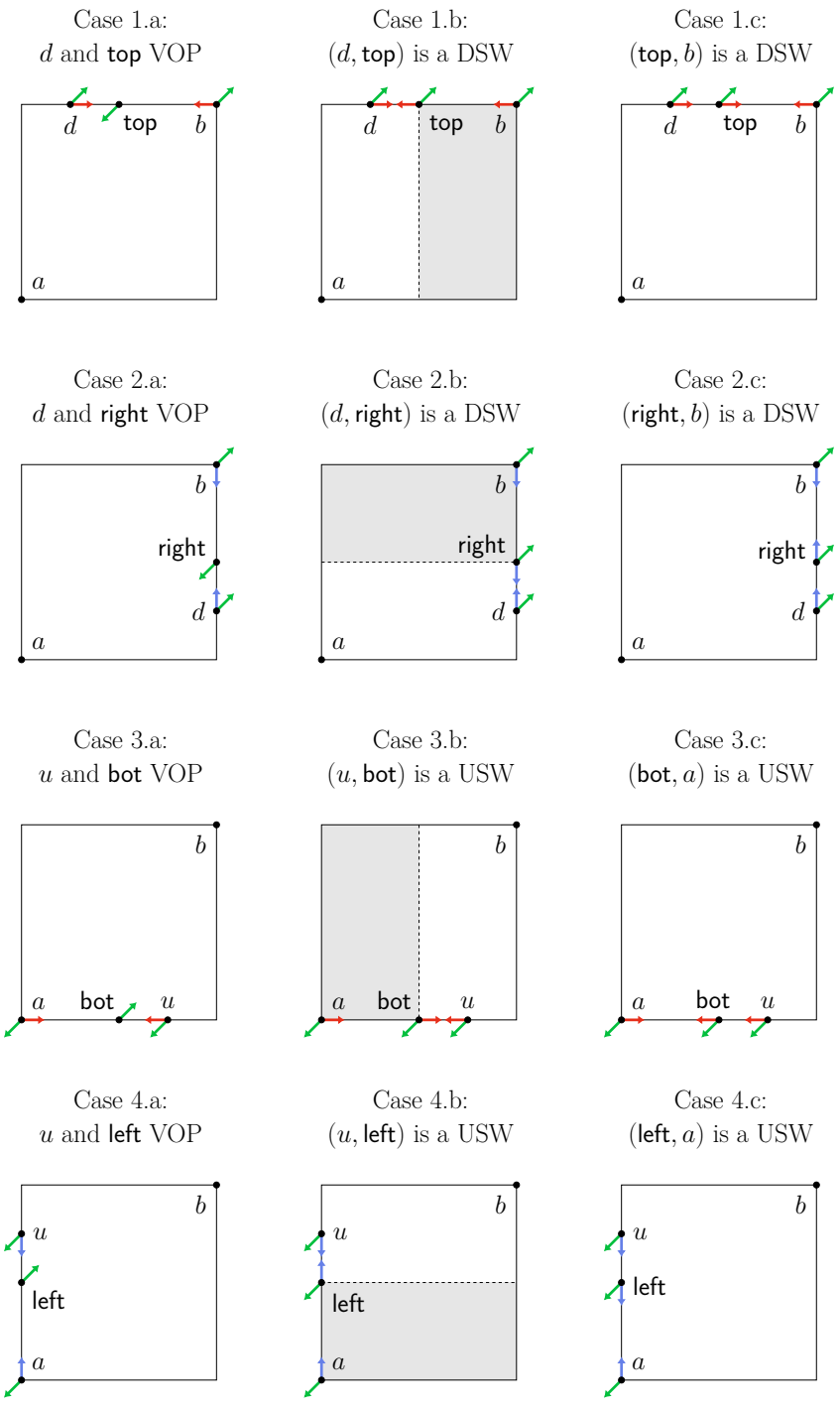

Figure 5 All cases used in Step 1 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

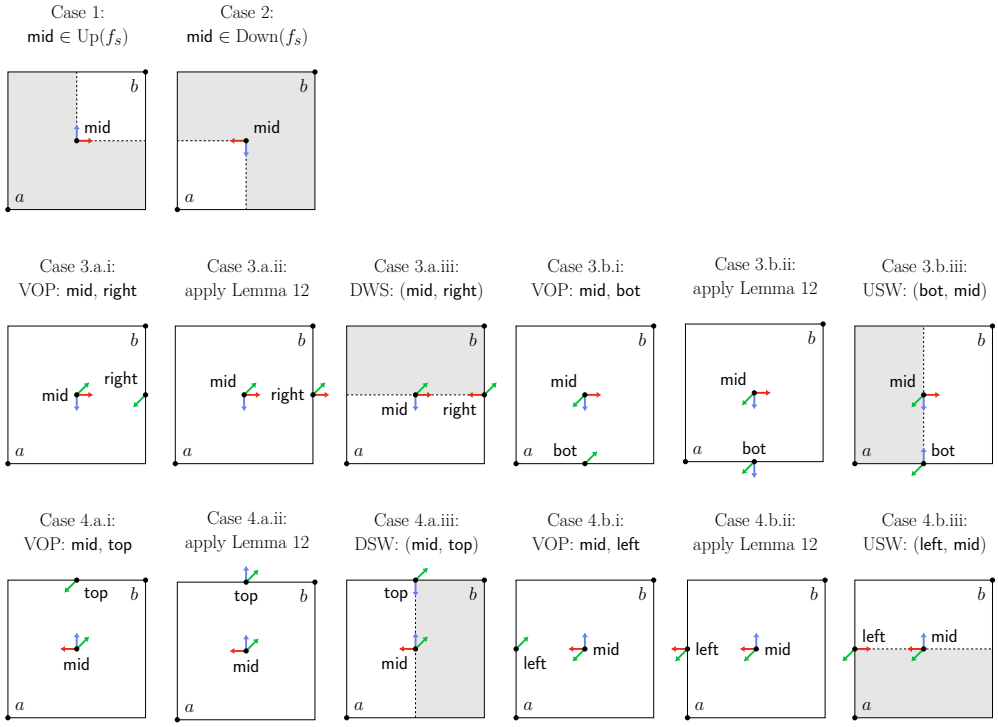

Figure 6 All cases used in Step 2 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

4. In this final case we have $\mathsf{mid}_1 > f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \leq f(\mathsf{mid})_2$. Here we follow the same procedure as Case 3, but with dimensions 1 and 2 exchanged, every instance of the point right replaced with top, and every instance of bot replaced with left.

The terminal phase of the algorithm. The algorithm can continue so long as $b_1 \geq a_1 + 2$ and $b_2 \geq a_2 + 2$, since this ensures that all cases will cut the width of one of the dimensions in half. The algorithm terminates once we have both $b_1 \leq a_1 + 1$ *and* $b_2 \leq a_2 + 1$. However, once there exists only one dimension i for which $b_i \leq a_i + 1$, we must be careful, since now the midpoint lies on the boundary of the instance, and some of the cases of the algorithm may not rule out anything. We deal with this scenario separately.

There are two distinct cases that we must deal with. The first case is a *width-one instance*, in which $b_i = a_i + 1$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$, meaning that the width of the shortest dimension is exactly one. These instances are problematic because the midpoint mid will now lie on the boundary of the instance, and due to this, it is possible that the algorithm may be unable to proceed.

We must also deal with *width-zero instances*, in which $b_i = a_i$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$. These are one-dimensional subinstances, and once again it is possible for the algorithm to be unable to proceed.

We will use special procedures for width-one and width-zero instances, which we outline below.

Width-one instances. In the presentation below we will assume that the index $i = 1$, meaning that $b_1 = a_1 + 1$ (and hence the left-right width of the instance is one). The case for $i = 2$ is symmetric.

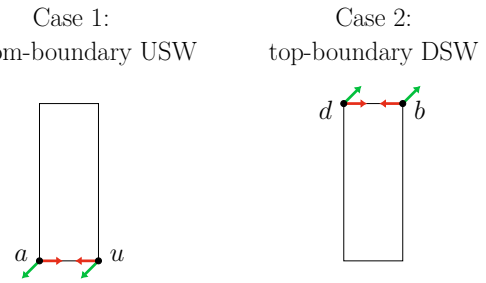

Figure 7 The two cases that trigger the preprocessing step for width-one instances.

When the algorithm is presented with a width-one instance, it first performs some preprocessing to ensure that there is no bottom-boundary up set witnesses, or top-boundary down set witness. The preprocessing considers the following two cases, which are shown in Figure 7.

1. If the instance has a bottom-boundary up set witness (a, u), then note that a and u are directly adjacent in dimension 1, and so Lemma 9 implies that we can either return a or u as a solution for the inner algorithm, or that $a \in \mathrm{Up}(f_s)$, or $u \in \mathrm{Up}(f_s)$.
   a. If $a \in \mathrm{Up}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the up set witness.
   b. If $u \in \mathrm{Up}(f_s)$, then the width-zero instance $L_{u,b}$ satisfies the invariant, where we note that if $L_{a,b}$ has a down set witness (d, b), then since $u \preceq d$, we have that (d, b) is also a valid down set witness for $L_{u,b}$.
2. If the instance has a top-boundary down set witness (d, b), then note that d and b are directly adjacent in dimension 1, and so Lemma 7 implies that we can either return d or b as a solution for the inner algorithm, or that $d \in \mathrm{Down}(f_s)$, or $b \in \mathrm{Down}(f_s)$.
   a. If $b \in \mathrm{Down}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the down set witness.
   b. If $d \in \mathrm{Down}(f_s)$, then the width-zero instance $L_{a,d}$ satisfies the invariant, where we note that if $L_{a,b}$ has an up set witness (a, u), then since $u \preceq d$, we have that (a, u) is also a valid up set witness for $L_{a,d}$.

With the preprocessing completed, the algorithm uses two separate runs of Steps 1 and 2, which each use a different midpoint. In the first run we use $\mathsf{midone} = \lfloor (a+b)/2 \rfloor$ as normal, while in the second run we use $\mathsf{midtwo} = \lceil (a+b)/2 \rceil$ as the midpoint, and we also change the definitions of bot, top, left, and right to round up instead of down. If either of the two runs decrease the size of the instance, then we move to the next iteration on the smaller instance, where the reasoning given in Steps 1 and 2 ensures that the instance continues to satisfy the invariant.

However, it could be the case that both runs do not decrease the size of the instance. Due to the preprocessing, if Step 1 attempts to recurse on a smaller sub-instance then it must succeed, since the only problematic cases are Case 1.b and Case 3.b, which both depend on the existence of a top or bottom-boundary witness, and the preprocessing ensures that these cannot exist.

On the other hand, Step 2 can fail to make progress in both runs. For the case where $i = 1$, this can only occur if Case 3.b.iii of Step 2 triggered for the run with midone and Case 4.a.iii triggered for the run with midtwo. But we can argue that in this case a solution to the inner algorithm is easy to find.

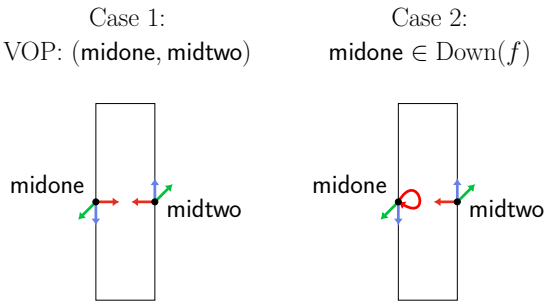

Figure 8 The two cases that are considered for width-one instances, when both runs of the algorithm fail to make progress. In the left instance, $f(\mathsf{midone})$ strictly increases in dimension 1; in the right instance, $f(\mathsf{midone})$ does not move in dimension one, which we indicate with the self loop.

Note that Case 3.b.iii can only trigger for midone when $\mathsf{midone}_1 \leq f(\mathsf{midone})_1$, while Case 4.a.iii can only trigger for midtwo when $f(\mathsf{midtwo})_1 < \mathsf{midtwo}$. Both of the following cases are shown in Figure 8.

1. If $\mathsf{midone}_1 < f(\mathsf{midone})_1$ then we have

$$f(\mathsf{midtwo})_1 \leq \mathsf{midtwo}_1 - 1 = \mathsf{midone}_1 < f(\mathsf{midone})_1,$$

   so we have $\mathsf{midone} \preceq \mathsf{midtwo}$ but $f(\mathsf{midone}) \not\preceq f(\mathsf{midtwo})$, meaning that midone and midtwo witness a violation of order preservation.
2. If $\mathsf{midone}_1 = f(\mathsf{midone})_1$, then note that Case 3.b.iii ensures that $\mathsf{midone}_2 \geq f(\mathsf{midone})_2$ and $\mathsf{midone}_3 \geq f(\mathsf{midone})_3$. Therefore midone is in $\mathrm{Down}(f)$, so can be returned by the inner algorithm.

So in both cases a solution to the inner algorithm has been found.

Width-zero instances. We again describe the procedure for the case where $i = 1$, meaning that the instance has width zero in the left-right dimension. The case where $i = 2$ is symmetric.

The algorithm begins by performing a preprocessing step that removes any top-boundary down set witnesses or bottom-boundary up set witnesses. If there is a bottom-boundary up set witness (a, u) then note that $a = u$, and therefore Lemma 9 implies that either a is a solution that can be returned by the inner algorithm, or that $a \in \mathrm{Up}(f_s)$. Likewise, if there is a top-boundary down set witness (d, b), then $d = b$, and Lemma 7 implies that either d can be returned by the inner algorithm, or $d \in \mathrm{Down}(f_s)$. Thus, the preprocessing step can either find a solution for the inner algorithm, or produce an instance that satisfies the invariant that has no top-boundary down set witness and no bottom-boundary up set witness.

Once the preprocessing has taken place, the algorithm proceeds through Step 1 and Step 2 as normal. If those steps make progress, then we continue on the smaller width-zero instance. If they do not make progress, then we will show that the inner algorithm can terminate after making at most $O(\log n)$ further queries.

We first observe that the only cases of Step 1 that would fail to make progress are Case 1.b and Case 3.b, but neither of those cases can trigger because the preprocessing step ensures that there is no bottom-boundary up set witness or top-boundary down set witness.

On the other hand, Case 3.b.iii and Case 4.a.iii of Step 2 can fail to make progress. We show how, in each of these cases, a solution for the inner algorithm can be found by making at most $O(\log n)$ extra queries. All of the following cases are depicted in Figure 9.

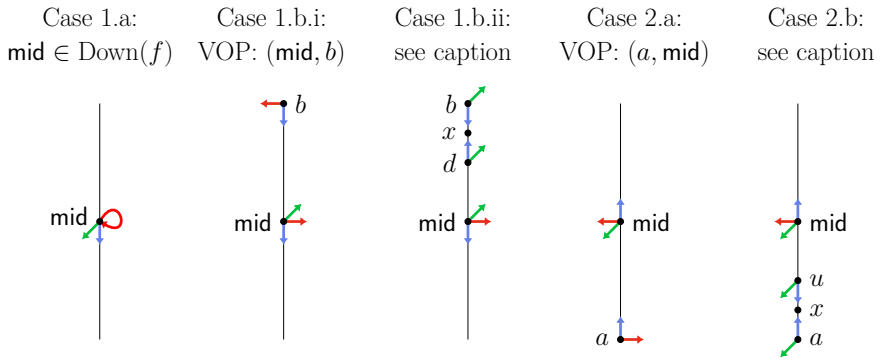

Figure 9 The five cases that can be encountered if the algorithm fails to make progress for a width-zero instance. In Cases 1.b.ii and 2.b we spend $O(\log n)$ queries to find the point x, which then allows us to terminate.

1. If Case 3.b.iii is triggered, then note that $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$. There are two cases to consider.
   a. If $\mathsf{mid}_1 = f(\mathsf{mid})_1$, then since $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$ and $\mathsf{mid}_3 \geq f(\mathsf{mid}_3)$, we have that $\mathsf{mid} \in \mathrm{Down}(f)$, meaning that mid can be returned by the inner algorithm.
   b. If $\mathsf{mid}_1 < f(\mathsf{mid})_1$ then we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
      i. If $b \in \mathrm{Down}(f_s)$, then we have

$$f(b)_1 \leq b_1 = \mathsf{mid}_1 < f(\mathsf{mid})_1,$$

      meaning that $\mathsf{mid} \preceq b$, but $f(\mathsf{mid}) \not\preceq f(b)$, and so mid and b violate order preservation.
      ii. If instead there is a down set witness (d, b), then note that due to the preprocessing, it must be a right-boundary down set witness, and due to Step 1, we must have $\mathsf{mid} \preceq d$. By Lemma 7 there exists a point x satisfying $d \preceq x \preceq b$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Down}(f_s)$. Furthermore, using we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Down}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.
2. If Case 4.a.iii is triggered, then note that $\mathsf{mid}_1 > f(\mathsf{mid})_1$, and we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
   a. If $a \in \mathrm{Up}(f_s)$, then we have

$$f(\mathsf{mid})_1 < \mathsf{mid}_1 = a_1 \leq f(\mathsf{mid})_1,$$

   meaning that $a \preceq \mathsf{mid}$, but $f(a) \not\preceq f(\mathsf{mid})$, and so a and mid violate order preservation.
   b. If instead there is an up set witness (a, u), then note that due to the preprocessing, it must be a left-boundary up set witness, and due to Step 1, we must have $u \preceq \mathsf{mid}$. By Lemma 9 there exists a point x satisfying $a \preceq x \preceq u$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Up}(f_s)$. Furthermore, we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Up}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.

Termination. If the algorithm does not hit any of the cases that return a solution immediately, then it will continue until it finds an instance $L_{a,b}$ with $b_1 \leq a_1 + 1$ and $b_2 \leq a_2 + 1$ that satisfies the invariant. Lemma 11 implies that any sub-instance that satisfies the invariant contains a solution that can be returned by the inner algorithm. Since then $L_{a,b}$ contains at most four points, we can check all of them and then return the solution that must exist.

Query complexity. Observe that each iteration of the algorithm either finds a violation of order preservation, finds a solution after spending $O(\log n)$ further queries, or reduces the size of one of the dimensions by a factor of two. Moreover, each non-terminating iteration of the algorithm queries at most five points. Hence, if the algorithm is run on a sub-instance $L_{a,b}$ with $n_1 = b_1 - a_1$ and $n_2 = b_2 - a_2$, then the algorithm will terminate after making at most $O(\log n_1 + \log n_2 + \log n)$ queries. So the overall query complexity of the algorithm is $O(\log n)$, and we have shown the following theorem.

▶ **Theorem 14.** *There is an $O(\log n)$-query inner algorithm for 3-dimensional* Tarski.

Theorems 5 and 14 imply that 3-dimensional Tarski can be solved using $O(\log^2 n)$ queries, and this can be combined with the $\Omega(\log^2 n)$ lower bound for two-dimensional Tarski [7], to give the following theorem.

▶ **Theorem 15.** *The deterministic query complexity of three-dimensional* Tarski *is $\Theta(\log^2 n)$.*

5 Extension to higher dimensions

We now extend our results to show that k-dimensional Tarski can be solved using $O(\log^{k-1} n)$ queries. The algorithm of Dang et al. [2] solves a k-dimensional Tarski instance by making $O(\log n)$ recursive calls to an algorithm for solving $(k-1)$-dimensional Tarski instances. Our algorithm can be plugged into this recursion as a new base case for $k = 3$.

The following lemma is a consequence of the work of Dang et al. [2]. However, their algorithm deals with the promise version of Tarski in which it is assumed that the input function is order preserving. For this reason, we provide our own proof of the lemma in which we give a variation of the algorithm that either finds a fixed point or explicitly provides a violation of order preservation.

▶ **Lemma 16.** *If $(k-1)$-dimensional* Tarski *can be solved using q queries, then k-dimensional* Tarski *can be solved using $(q+2) \cdot (\log n + 2)$ queries.*

The direct consequence of Lemma 16 and our $O(\log^2 n)$ query algorithm for three-dimensional Tarski is the following theorem.

▶ **Theorem 17.** *k-dimensional* Tarski *can be solved using $O(\log^{k-1} n)$ queries for $k \geq 3$,*

Time complexity. To obtain time complexity results, note that writing down a point in the lattice L already requires $k \cdot \log n$ time. We assume that f is implemented by a Boolean circuit of size that is polynomial in k and $\log n$. With this assumption, our time complexity result can be stated as follows.

▶ **Theorem 18.** *If f is presented as a Boolean circuit of size* $\text{poly}(\log n, k)$*, then for $k \geq 3$ there is an algorithm for* Tarski *that runs in time $O(\text{poly}(\log n, k) \cdot \log(n)^{k-1})$.*

6 Conclusion

Our $O(\log^{k-1} n)$ query algorithm for k-dimensional Tarski falsifies prior conjectures that the problem required $\Omega(\log^k n)$ queries [2, 7]. This, of course, raises the question of what is the query complexity of finding a Tarski fixed point? While our upper bound is tight in three dimensions, it seems less likely to be the correct answer in higher dimensions. Indeed, there seems to be a fairly wide range of possibilities. Is it possible to show a $\log^{\Omega(k)} n$ query lower bound for the problem? Or perhaps there exists a fixed parameter tractable algorithm that uses $O(f(k) \cdot \log^2 n)$ queries? Both of those would be consistent with the known upper and lower bounds, and so further research will be needed to close the gap.

References

1 Ching-Lueh Chang, Yuh-Dauh Lyuu, and Yen-Wu Ti. The complexity of Tarski's fixed point theorem. *Theor. Comput. Sci.*, 401(1-3):228–235, 2008.

2 Chuangyin Dang, Qi Qi, and Yinyu Ye. Computations and complexities of Tarski's fixed points and supermodular games. *CoRR*, abs/2005.09836, 2020. Stanford tech report version appeared in 2012. `arXiv:2005.09836`.

3 Chuangyin Dang and Yinyu Ye. On the complexity of a class of discrete fixed point problems under the lexicographic ordering. Technical report, City University of Hong Kong, 2018. CY2018-3, 17 pages.

4 Chuangyin Dang and Yinyu Ye. On the complexity of an expanded tarski's fixed point problem under the componentwise ordering. *Theor. Comput. Sci.*, 732:26–45, 2018.

5 Chuangyin Dang and Yinyu Ye. Erratum/correction to "on the complexity of an expanded tarski's fixed point problem under the componentwise ordering" [Theor. Comput. Sci. 732 (2018) 26-45]. *Theor. Comput. Sci.*, 817:80, 2020.

6 Federico Echenique. Finding all equilibria in games of strategic complements. *J. Econ. Theory*, 135(1):514–532, 2007.

7 Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's theorem, supermodular games, and the complexity of equilibria. In *Proc. of ITCS*, volume 151, pages 18:1–18:19, 2020.

8 John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. *CoRR*, abs/2011.01929, 2020. To appear in the Proccedings of the 53nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021. `arXiv:2011.01929`.

9 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *J. Comput. Syst. Sci.*, 114:1–35, 2020.

10 Paul Milgrom and John Roberts. Rationalizability, learning, and equilibrium in games with strategic complementarities. *Econometrica*, 58(6):1255–1277, 1990.

11 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

12 Donald M. Topkis. Equilibrium points in nonzero-sum n-person submodular games. *SIAM J. Control Optim*, 17:773–787, 1979.

13 Donald M. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.