A Appendix: Decision Tree

In this section, we introduce the connection between optimal question selection problem and optimal decision tree problem, and thus complete the proofs of the computation complexity of $\mathcal{OQS}$ and approximation ratio of *minimax branch* given in Section 2.

A.1 Definitions

Definition A.1. (Decision tree) Given a set of m tests T, a set of n items X, and a set of test results R, where each pair of $t \in T$ and $x \in X$ is associated with a test result $r \in R$, denoted as $t(x)$. A *decision tree DT* on T, X is a rooted tree that satisfies the following properties:

- Each internal node is labeled by a test in T.
- Each internal node has at least two children.
- Each outgoing edge of each internal node is labeled by a value in R.
- For any internal node, the labels of its outgoing edges are different from each other.
- Each leaf node is labeled by a non-empty subset of X.
- Given a path from the root to a leaf $(t_1, r_1, \dots, t_n, r_n, X_0)$, we have $t_i(x) = r_i$ for any $1 \le i \le n$ and $x \in X_0$.
- For any two $x_1, x_2 \in X$, x_1 and x_2 are labeled on the same leaf iff for any $t \in T$, $t(x_1) = t(x_2)$.

Example A.2. Figure 5 shows an example of decision trees. The items are eight integers $\{0, 1, 6, 11, 12, 15, 16, 21\}$, and the tests are calculating the remainder of dividing 2, 3, 5 respectively. The left tabular lists the corresponding test results. A decision tree of these tests and items is shown on the right side. Each internal node is marked with its label and each leaf is marked with its corresponding item.

		T_1	T_2	T_3
		mod 2	mod 3	mod 5
X_1	0	0	0	0
X_2	1	1	1	1
X_3	6	0	0	1
X_4	11	1	2	1
X_5	12	0	0	2
X_6	15	1	0	0
X_7	16	0	1	1
X_8	21	1	0	1

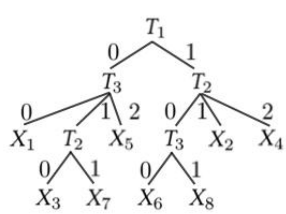

Figure 5. An example of decision trees. The right tree is a decision tree constructing from the items and tests which are listed in the left tabular.

Given a decision tree DT, we define the cost $c(X_i)$ of an item X_i be the depth of its corresponding leaf (the depth of the root is defined as 0). This definition has an explicit realistic correspondence: it is equal to the number of the tests used by DT to identify X_i. For instance, on the decision tree shown in Figure 5, $c(X_1) = c(X_2) = c(X_4) = C(X_5) = 2, c(X_3) = c(X_6) = c(X_7) = c(X_8) = 3$.

Based on the definitions of decision trees and the costs of items, we now describe the problem of constructing an optimal decision tree.

Definition A.3. ($\mathcal{DT}$ Problem) Given a set of tests T, a set of items X, a set of test results R, and a probability distribution φ on X, which assigns a probability to each item in X and all probabilities sum up to 1, the goal of *optimal decision tree problem* $\mathcal{DT}$ is to construct a decision tree in which the expected cost of a random item is minimized, i.e., find a decision tree that minimizes $\sum_{x \in X} \varphi(x)c(x)$. This expected cost is called as the *cost of a decision tree.*

Specially, when φ is fixed to a uniform distribution on X, the problem is named as $\mathcal{UDT}$. When the tests are all binary, i.e., the results are always Boolean values, the problem is named as $2-\mathcal{DT}$. Besides, when the two conditions are satisfied at the same time, the name is $2-\mathcal{UDT}$.

Example A.4. In Example A.2, when φ is a uniform distribution on X, the cost of the decision tree in Figure 5 is $\frac{5}{2}$. When φ is a uniform distribution on $\{X_1, X_5, X_6\}$ and other items have no chance to occur, the cost of the decision tree is $\frac{7}{3}$. We show the corresponding optimal decision trees in Figure 6. The left tree is the answer of $\mathcal{UDT}$ with the minimum cost $\frac{19}{8}$, and the right tree is the answer of $\mathcal{DT}$ with the minimum cost $\frac{5}{3}$ when q is a uniform distribution on $\{X_1, X_5, X_6\}$.

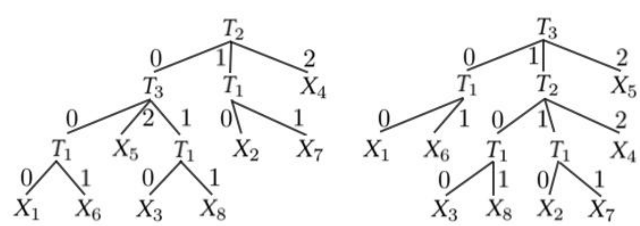

Figure 6. The left tree and the right tree are the optimal decision tree when q is a uniform distribution on X and on $\{X_1, X_5, X_6\}$ respectively.

A.2 Some results on constructing optimal decision trees

Complexity of Optimal Decision Tree. As decision trees are used in many fields, constructing an optimal decision tree has become an important and fascinating subject. Over the past decades, there have been a lot of theoretical results and algorithms published. As many other optimization problems, $2-\mathcal{UDT}$ was proved as an NP-complete problem in very early days [28], which implies $\mathcal{UDT}$, $2-\mathcal{DT}$, $\mathcal{DT}$ are all NP-hard. After realizing the hardness of finding an optimal polynomial-time algorithm, most of subsequent works focus on finding efficient approximation algorithms and proving the hardness of approximating $\mathcal{DT}$.

An Approximation Algorithm. As the background, we first give the definitions of approximation algorithms, approximation ratios and the hardness of approximation. Intuitively, the approximation ratio measures the distance between the optimal solution and the solution given by a approximation algorithm, and the hardness of approximation shows how hard to find an approximation algorithm within a given approximation ratio.

Definition A.5. (Approximation Algorithm) For an optimization problem $\mathcal{P}$ of which the goal is to minimize an objective function c, an algorithm A is an approximation algorithm with approximation ratio k iff it satisfies the following two conditions:

- For any instance of $\mathcal{P}$, the result of A is always valid.
- For any instance of $\mathcal{P}$, the inequality $c(S) \leq kc(S')$ holds for any valid solution S', where S is the solution given by A.

Definition A.6. (Hardness of Approximation) An optimization problem $\mathcal{P}$ is hard to be approximated within approximation ratio k iff there is no polynomial-time algorithm that approximates $\mathcal{P}$ within ratio k unless $P = NP$.

With these definitions, we show how *minimax branch* works for $\mathcal{DT}$. For each node, *minimax branch* always choose the test which partition the items represented by this node as equally as possible. More concretely, we assume there is a function *par* that partitions a set of items S based on a test $t_i \in T$, i.e., $par(S, t_i) = \{S'_1, \dots, S'_k\}$ where each S_i corresponds to a possible result of the test. We further define the cost of test t_i on item set S as $cost(t_i, S) = \max_{S'_j \in par(S,t_i)} \sum_{x \in S'_j} \varphi(x)$, i.e., the largest sum of probabilities among all subsets in this partition. Then *minimax branch* always chooses the test with the smallest cost to partition the items.

Many research results indicate that *minimax branch* is an efficient approximation algorithm. We introduce them below.

Approximation Ratios of *minimax branch*. Table 3 summaries the state-of-the-art results in different branches of $\mathcal{DT}$ problem. As we can see from the table, *minimax branch* is the state-of-the-art algorithm for the first three problems. The only exception is $\mathcal{DT}$, of which a polynomial time algorithm with an ratio $O(\log m)$ is found [21]. However, that algorithm is much more complex than *minimax branch*.

On the other hand, under the assumption that $P \neq NP$, $2 - \mathcal{UDT}$ and $\mathcal{UDT}$ are hard to be approximated within any factors less than 2 and 4 respectively, and there is an approximation ratio $r(m) = \Omega(\log m)$ within which both $2 - \mathcal{DT}$ and $\mathcal{DT}$ are hard to be approximated. Especially, for $2 - \mathcal{DT}$, the approximation ratio $O(\log m)$ of *minimax branch* has already reached the lower bound $\Omega(\log m)$. Therefore, *minimax branch* is an optimal polynomial time algorithm to approximate $2 - \mathcal{DT}$ unless $P = NP$.

Problem	Hardness	Ref.	*minimax branch*	Ref.
$2 - \mathcal{UDT}$	$2 - \epsilon, \forall \epsilon > 0$	[10]	$1 + \ln m$ *	[1]
$\mathcal{UDT}$	$4 - \epsilon, \forall \epsilon > 0$		$O(\log m)$ *	[11]
$2 - \mathcal{DT}$	$\Omega(\log m)$		$O(\log m)$ *	[10]
$\mathcal{DT}$	$\Omega(\log m)$		$O(\log^2 m)$	

* The best approximation ratio found in polynomial time so far.

Table 3. The summary of related results. The second column shows the hardness of approximation and the fourth column shows the proved approximation ratio of the *minimax branch* strategy. Parameter m represents the number of items, i.e., $|X|$.

A.3 Relationship between Question Selection and Decision Tree

In this subsection, we discuss the relationship between optimal question selection problem and optimal decision tree problem. Intuitively, each program in the program space corresponds to an item to be distinguished by the decision tree, each question corresponds to a test, and each answer corresponds to a test result.

From optimal decision tree to question selection. We first consider how to reduce optimal decision tree problem to optimal question selection problem. We start by defining a mapping from instances of $\mathcal{DT}$ to instances of $\mathcal{OQS}$. More concretely, we have the following mapping.

$$\pi(X, T, R, \varphi) = (\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi')$$

where

- X is a set of items.
- T is a set of tests.
- R is a set of test results.
- φ is a distribution over items in X.
- $\mathbb{P} = \{\pi_x(t) \mid x \in X\}$ is a set of programs with if-else structures. Let $t_1, t_2, \dots, t_n$ be items in T. The program $\pi_x(t)$ for item x is defines as follows.

```
// program for x
if input == t_1 then return test result of t_1 on x
else if input == t_2 then return test result of t_2 on x
...
else if input == t_{n-1} then return test result of t_{n-1} on x
else return test result of t_n on x
```

- $\mathbb{Q}$ is a question domain and is equal to T.
- $\mathbb{A}$ is a answer domain and is equal to R.
- φ' is a distribution over programs in $\mathbb{P}$ and $\varphi'(\pi_x(t))$ is defined as $\varphi(x)$.

It is easy to see that π can be implemented as a polynomial-time algorithm.

We then present Algorithm 4 which constructs a decision tree from a question selection function QS. The algorithm calls Construct with an empty question-answer sequence (line 14). The function Construct first calls the question selection function QS (line 3) and creates either a leaf node (line 4) or an internal node (lines 6-9) based on whether the return value is $\top$ or not. When creating an internal node, the children are created by recursively calling Construct with the chosen question and the corresponding question-answer sequence.

Algorithm 4 Constructing a decision tree using a question selection function

Input: A set of tests T, a set of items X, a set of results R, a probability distribution φ on X and a question selection function QS.

Output: The root node of a decision tree.

1: **function** Construct($\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi', C$)
2:     *node* $\leftarrow$ NewNode
3:     **if** $QS(C) = \top$ **then**
4:         The label of *node* $\leftarrow \mathbb{P}|_C$
5:     **else**
6:         $q^* \leftarrow QS[\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi'](C)$
7:         The label of *node* $\leftarrow q^*$
8:         $A \leftarrow \{\mathbb{D}[p](q^*) \mid p \in \mathbb{P}|_C\}$
9:         Children of *node*$\leftarrow\{$NewChild$(o,$
10: Construct$(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi', C \cup \{(q^*, a)\})) \mid a \in A\}$
11:     **end if**
12:     **return** node
13: **end function**
14: $(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi') \leftarrow \pi(X, T, R, \varphi)$
15: **return** Construct$(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi', ())$

Lemma A.7. *Algorithm 4 returns a decision tree.*

Proof. Obviously, Algorithm 4 returns a tree. To show it is a decision tree, we need to check whether the properties of a decision tree are satisfied or not. First, it is easy to see the internal nodes, the edges, and the leaf nodes are labeled correctly, and the outgoing edges of the same node have different labels. Second, because QS returns a non-$\top$ value only when some question can further distinguish remaining programs, each internal node must have at least two children. Third, since the label of a leaf node is the set of valid programs with respect to C, items labeled on the leaf node must be consistent with previous tests. Fourth, since QS returns $\top$ only when the remaining programs cannot be distinguished by any question-answer pairs, we know that items on the same leaf cannot be further distinguished by any tests. Fifth, since the tree construction process considers all items and any two leaves have different paths from the root, any two indistinguishable items must be labeled on the same leaf node. □

Lemma A.8. *Given an item $x \in X$, the path from the root to x on the decision tree returned by Algorithm 4 has the length of $len(QS, x)$.*

Proof. Given an item x, iteratively calling QS generates a sequence of questions. It is easy to see len returns the length of this sequence. By induction on the first n questions in the sequence, we can see that (1) there is always a path corresponding to the first n questions; (2) when QS returns $\top$, the path reaches a leaf node whose labeled set must contains x. Putting them together, we know the lemma holds. □

Lemma A.9. *Algorithm 4 has polynomial time complexity if QS has polynomial time complexity.*

Proof. First, each call to Construct creates a new node, so the number of calls to Construct is equal to the size of the decision tree, which is a polynomial of the number of items in X. Second, when $QS[\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi']$ is polynomial-time, each statement in Construct costs polynomial time with respect to the size of X and the size of T. As a result, the whole algorithm is polynomial-time. □

Based on these lemmas, we know that $\mathfrak{DT}$ is polynomial-time reducible to $\mathfrak{OQS}$, and thus we can adapt several results from previous research works on $\mathfrak{DT}$.

Theorem A.10. *The problem of approximating $\mathfrak{DT}$ with approximation ratio k is polynomial-time reducible to the problem of approximating $\mathfrak{OQS}$ with approximation ratio k.*

Proof. This is a direct result of Lemma A.8, Lemma A.9, and Lemma A.7. □

Corollary A.11. *$\mathfrak{DT}$ is polynomial-time reducible to $\mathfrak{OQS}$.*

Proof. Let k be 1. Theorem A.10 reduces to this corollary. □

Corollary A.12 (Theorem 2.6)**.** *$\mathfrak{OQS}$ is NP-hard.*

Proof. By contradiction, if $\mathfrak{OQS}$ is not NP-hard, $\mathfrak{DT}$ cannot be NP-hard by Theorem A.10. □

Let $2-\mathfrak{OQS}$ be the subproblem of $\mathfrak{OQS}$ when all questions are boolean, i.e., $|\mathbb{A} = 2|$, $\mathfrak{UOQS}$ be the subprogram when φ is a uniform distribution and $2-\mathfrak{UOQS}$ be the subprogram when both two conditions are met.

Corollary A.13. *$\mathfrak{OQS}$ and $2-\mathfrak{OQS}$ are hard to be approximated with a ratio of $\Omega(\log|\mathbb{P}|)$; $\mathfrak{UOQS}$ is hard to be approximated with a ratio of $4-\epsilon$ for any $\epsilon > 0$; $2-\mathfrak{UOQS}$ is hard to be approximated with a ratio of $2-\epsilon$ for any $\epsilon > 0$.*

Proof. Similar to previous corollaries, this corollary can be proved by contradiction. □

From question selection to optimal decision tree. Now we show the other direction is also polynomial-time reducible. Similarly, we first define a mapping from instances of $\mathfrak{OQS}$

to instances of $\mathcal{DT}$. More concretely, we construct the following mapping.

$$\rho(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi') = (X, T, R, \varphi)$$

where

- $\mathbb{P}$ is a domain of programs.
- $\mathbb{Q}$ is the question domain $\mathbb{P}$ defined on.
- $\mathbb{A}$ is the answer domain $\mathbb{P}$ defined on.
- φ' is the distribution over programs in $\mathbb{P}$.
- X is a set of items to be distinguished by the decision and is equal to $\mathbb{P}$.
- T is a set of tests and is equal to $\mathbb{Q}$, where the test result of $q \in \mathbb{Q}$ on $x \in X$ is equal to $\mathbb{D}[x](q)$.
- R is a set of test results and is equal to $\mathbb{A}$.
- φ is a distribution over items in X and is equal to φ'.

It is easy to see that ρ can be implemented as a polynomial-time algorithm if programs in $\mathbb{P}$ are all polynomial-time. To better discuss the complexity of $\mathcal{OQS}$, we ignore the time complexities of programs in $\mathbb{P}$ and assume $\mathcal{D}$ can always be calculated within constant time.

Based on ρ, we present Algorithm 5, which solves the question selection problem with a decision tree constructor Construct. This algorithm directly constructs a decision tree, and returns the label of the corresponding path. We have the following three lemmas for this algorithm.

Algorithm 5 Constructing a question selection function using a decision tree constructor

Input: A program space $\mathbb{P}$, a question domain $\mathbb{Q}$, an answer domain $\mathbb{A}$, a probability distribution φ of $\mathbb{P}$, and a sequence of question-answer pairs C
Output: the next question .

```
(X, T, R, φ') ← ρ(ℙ, ℚ, 𝔸, φ)
d ← Construct(X, T, R, φ')
if there is a path from the root in d corresponding to C then
    path ←the path in d corresponding to C
else
    return ⊤
end if
node ←the end of path
if node is a leaf node then
    return ⊤
else
    return the label of node
end if
```

Lemma A.14. *Algorithm 5 defines a question selection function.*

Proof. We need to show two properties of an input selection function hold. The first one is a direct result of the last property of a decision tree. The second one is ensured since each internal node has at least two children. □

Lemma A.15. *Let the question selection function defined by Algorithm 5 be QS. Given a program $p \in \mathbb{P}$, the path from the root to p in the decision tree returned by* Construct *has the length of $len(inSel, p)$.*

Proof. This can be proved by showing that the algorithm always traverses exactly this path during each invocation. □

Lemma A.16. *Algorithm 5 is polynomial-time if both* Construct *and all programs in $\mathbb{P}$ are polynomial-time.*

Proof. It is easy to see there is no loop in the algorithm and each statement costs polynomial time if the preconditions are satisfied. □

Theorem A.17. *The problem of approximating $\mathcal{OQS}$ with approximation ratio k is polynomial-time reducible to the problem of approximating $\mathcal{DT}$ with approximation ratio k.*

Proof. This is a direct result from Lemma A.16, Lemma A.14, and Lemma A.15. □

Corollary A.18. *$\mathcal{OQS}$ is polynomial-time reducible to $\mathcal{DT}$.*

Proof. Let k be 1. Theorem A.17 reduces to this corollary. □

Now, we can apply above results to show that *minimax branch* is an efficient approximation algorithm for $\mathcal{OQS}$ and its subproblems.

Theorem A.19 (Theorem 2.8)**.** *The minimax branch strategy for $\mathcal{OQS}$ problem has the approximation ratio $1 + \ln m$ for $2 - \mathcal{UOQS}$,$O(\log m)$ for $\mathcal{UOQS}$ and $2 - \mathcal{OQS}$, and $O(\log^2 m)$ for $\mathcal{OQS}$, where $m = |\mathbb{P}|$.*

Proof. This corollary can be proved as follows. We first implement a question selection function by converting *minimax branch* on $\mathcal{DT}$ through Algorithm 5. Clearly, this converted algorithm has the same approximation ratios on $\mathcal{OQS}$ as on $\mathcal{DT}$. Furthermore, by the definition of *minimax branch* on $\mathcal{OQS}$, it always chooses the same input as the above converted question selection function. Therefore, we obtain the approximation ratios of *minimax branch* on $\mathcal{OQS}$ and its subproblems. □

B Appendix: Proofs

In this section, we complete the proofs of the theorems in Section 3, 4 and 5.

Lemma B.1 (Lemma 3.1)**.** *For any $\mathcal{OQS}$ instance $(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi)$ and any $P \subseteq \mathbb{P}$, let q_0 and q_1 be the questions selected by* Minimax0 *and* Minimax*, respectively, then q_0 and q_1 have the same efficiency on P, i.e.:*

$$\max_{a \in \mathbb{A}} \left| P|_{(q_0, a)} \right| = \max_{a \in \mathbb{A}} \left| P|_{(q_1, a)} \right|$$

Proof. By the definition of $P|_{(i,o)}$, we have:

$$\forall q \in \mathbb{Q},\ \max_{a \in \mathbb{A}} \left| P|_{(q,a)} \right| = \max_{j} \left(\sum_{k=j+1}^{n} [\mathbb{D}[p_j](q) = \mathbb{D}[p_k](q)] \right)$$

Apply this equality to Minimax, and then we obtain this lemma. □

Theorem B.2 (Theorem 3.2). *For any $\mathcal{OQS}$ instance $(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi)$ and any sequence of examples $C \in (\mathbb{Q} \times \mathbb{A})^*$, define the cost of a question cost(q) to be $\max_{a\in\mathbb{A}} w\left(\mathbb{P}|_{C\cup\{(q,a)\}}\right)$. Let P be a set of independent samples from distribution $\varphi|_C$, q_1 and q_0 be the questions chosen by SampleSy and minimax branch respectively, then $\forall \epsilon > 0$:*

$$\Pr\left[cost(q_1) > (1+\epsilon)cost(q_0)\right] \leq 2d|\mathbb{Q}| \exp\left(-\frac{|P|\epsilon^2}{2d^2}\right)$$

where $d = \max_{q\in\mathbb{Q}} |\{\mathbb{D}[p](q) \mid p \in \mathbb{P}\}|$ which represents the maximum number of different answers on a single question.

Proof. Let n be the number of samples and $P = \{p_1, \dots, p_n\}$ be the samples. By the definition of the sampler, $p_1, \dots, p_n$ are independent samples from distribution $\varphi|_C$. For a question $q \in \mathbb{Q}$, answer $a \in \mathbb{A}$, let $f(q,a) = \Pr_{p\sim(\varphi|_C)}[\mathbb{D}[p](q) = a]$, i.e., the probability for a sample p to be consistent with question-answer example (q,a).

Then, by Chernoff bound:

$$\forall \epsilon' > 0, q \in \mathbb{Q}, a \in \mathbb{A}, \Pr\left[\left|\frac{\left|P|_{(q,a)}\right|}{n} - f(q,a)\right| > \epsilon'\right] < 2e^{-2n\epsilon'^2}$$

Take union bound on all possible inputs and corresponding outputs, we obtain:

$$\forall \epsilon' > 0, \Pr\left[\exists q \in \mathbb{Q}, \exists a \in \mathbb{A}, \left|\frac{\left|P|_{(q,a)}\right|}{n} - f(q,a)\right| > \epsilon'\right] < 2d|\mathbb{Q}|e^{-2n\epsilon'^2}$$

For a question $q \in \mathbb{Q}$, we define two cost functions:

- $c_1(q) = \max_{a\in\mathbb{A}} \frac{|P|_{(q,a)}|}{n}$, which represents the cost on the samples.
- $c_2(q) = \max_{a\in\mathbb{A}} f(q,a)$, which represents the cost on $\mathbb{P}|_C$. Note that $c_2(q) \times w(\mathbb{P}|_C) = cost(q)$

By the definitions of q_0 and q_1, we know that q_0 and q_1 are $\arg\min_{q\in\mathbb{Q}} c_2(q)$ and $\arg\min_{q\in\mathbb{Q}} c_1(q)$, respectively.

Notice that:

$$\begin{aligned}
&\forall q \in \mathbb{Q}, \forall a \in \mathbb{A}, \left|\frac{\left|P|_{(q,a)}\right|}{n} - f(q,a)\right| \leq \epsilon' \\
&\implies \forall q \in Q, |c_1(q) - c_2(q)| \leq \epsilon' \\
&\implies c_2(q_1) - c_2(q_0) \leq 2\epsilon'
\end{aligned}$$

The last inequality holds because:

$$\begin{aligned}
&c_2(q_1) - c_2(q_0) \\
\leq\ &(c_1(q_1) - c_1(q_0)) + |c_1(q_1) - c_2(q_1)| + |c_1(q_0) - c_2(q_0)|
\end{aligned}$$

Consider the three terms on the right side: Since q_1 is the best input with respect to P, the first term is non-positive. With the precondition that for all inputs, the difference between c_1 and c_2 are bounded, the second and the third term are both no more than ϵ'. Therefore the sum is at most $2\epsilon'$.

Take ϵ' as $\frac{1}{2}\epsilon$, we obtain:

$$\forall \epsilon > 0, \Pr\left[c_2(q_1) - c_2(q_0) > \epsilon\right] \leq 2|\mathbb{Q}||\mathbb{A}|e^{-\frac{1}{2}n\epsilon^2}$$

The definition of $c_2(q) = \max_{a\in\mathbb{A}} f(q,a)$ implies $c_2(q) \geq \frac{1}{d}$. Substitute ϵ with $\epsilon c_2(q_0)$:

$$\begin{aligned}
\Pr\left[c_2(q_1) > c_2(q_0)(1+\epsilon)\right] &\leq 2d|\mathbb{Q}| \exp\left(-\frac{|P|\epsilon^2 c_2(q_0)^2}{2}\right) \\
&\leq 2d|\mathbb{Q}| \exp\left(-\frac{|P|\epsilon^2}{2d^2}\right)
\end{aligned}$$

Because $c_2(q)$ is proportional to the value of $cost(q)$, this inequality is equivalent to the theorem □

Lemma B.3 (Lemma 4.5). *For any program set P and constant w, define $B(P) \subseteq P$ as the set of programs which makes $\psi_{good}[p](q,w)$ unsatisfiable, then:*

- *$w \leq 1/2 \implies \forall P$, programs in $B(P)$ are indistinguishable from each other.*
- *$w > 1/2 \implies \forall t > 0, \exists P$ s.t. $B(P)$ contains at least t programs which are distinguishable from each other.*

Proof. For the first part, let p_1, p_2 be any two programs in $B(P)$ and q be a question in $\mathbb{Q}$, let $P_1(q), P_2(q)$ be the set of programs in P which have the same answer as p_1 and p_2 on q, respectively.

By the definition of $B(P)$, $|P_1(q)| > \frac{|P/p_1|}{2} + n(p_1)$, $|P_2(q)| > \frac{|P/p_2|}{2} + n(p_2)$ where $n(p)$ represents the number of p in set P. Therefore $|P_1(q)|, |P_2(q)| > \frac{|P|}{2}$, which implies $|P_1(q) \cap P_2(q)| > 0$. By the definition of $P_1(q)$ and $P_2(q)$, we have $\mathbb{D}[p_1](q) = \mathbb{D}[p_2](q)$. Since p_1, p_2, q are chosen arbitrarily, we obtain that all programs in $B(P)$ are indistinguishable.

For the second part, we can construct such a P in the following way:

- let m be a large enough integer which satisfies $2^m \geq t$ and $\frac{2^{m-1}}{2^m-1} < w$.
- P contains 2^m programs, $\mathbb{Q}$ contains m questions, and the answer for the ith program on the jth question is equal to the jth digit in the binary representation of i.

Then, for any $p \in P, q \in \mathbb{Q}$, $|P/p| = 2^m - 1$ and there are 2^{m-1} programs performing differently with p. Since $\frac{2^{m-1}}{2^m-1} < w$, $B(P)$ contains all the programs in P, which are all distinguishable from each other. □

Theorem B.4 (Theorem 4.6). *For an $\mathcal{OQS}$ instance $(\mathbb{P}, \mathbb{Q}, \mathbb{A}, \varphi, \epsilon)$, let n be a lower bound of the number of samples in each turn, β be an upper bound of questions required by EpsSy, then:*

$$\begin{aligned}
\forall n > \max\left(18\ln\left(\frac{2\beta|\mathbb{Q}|}{\epsilon}\right), \frac{16\ln 2}{\epsilon^2} + \frac{8}{\epsilon^2}\ln\left(\frac{1}{\epsilon}\right)\right), \\
\forall f_\epsilon > \log_{3/2}\left(\frac{2\beta}{\epsilon}\right), e(EpsSy, p) \leq \epsilon
\end{aligned}$$

Proof. There are two termination conditions in *EpsSy*. Let $e_1(EpsSy, \varphi)$ be the error rate when *EpsSy* returns through condition one (Line 5 in Algorithm 2) and $e_2(EpsSy, \varphi)$ be that for condition two (Line 16 in Algorithm 2). The theorem is equivalent to both e_1 and e_2 are bounded.

For $e_1(EpsSy, \varphi)$, let C be the final question-answer sequence, m be the number of samples which are indistinguishable from p^*, $ind(p^*)$ be the probability for a random program from distribution $\varphi|_C$ to be indistinguishable from p^*. By Chernoff bound:

$$\forall \epsilon' > 0, \Pr\left[\frac{m}{n} - ind(p^*) > \epsilon'\right] \leq \exp(-2n\epsilon'^2)$$

Since $\frac{m}{n} > 1 - \frac{\epsilon}{2}$, take ϵ' as $\frac{\epsilon}{4}$, then:

$$\Pr\left[ind(p^*) < 1 - \frac{3}{4}\epsilon\right] \leq \exp(-n\epsilon^2/8)$$

Since the target program also follows the conditional distribution $\varphi|_C$, the error rate at this time should be $1 - ind(p^*)$. Combining with above inequality, we obtain:

$$\begin{aligned} e_1(EpsSy, \varphi) &\leq \frac{3}{4}\epsilon \Pr[ind(p^*) \geq 1 - \frac{3}{4}\epsilon] + \Pr[ind(p^*) < 1 - \frac{3}{4}\epsilon] \\ &\leq \frac{3}{4}\epsilon + \exp(-n\epsilon^2/8) \end{aligned}$$

Then for $e_2(EpsSy, \varphi)$, let r_j, q_j be the recommendation and the question of the jth round, C_j be the question-answer sequence before the jth round, and r^* be the target program. Note that r^* is a random variable which is subject to φ, and its distribution conditional on C_j is $\varphi|_{C_j}$. For convenience, we involve two properties of interaction rounds:

- The jth round is *good* iff the question q_j is challengeable.
- The jth round is *effective* iff the following inequality holds:

$$\Pr_{r \sim \varphi|_{C_j}}\left[\mathbb{D}[r^*](q_j) \neq \mathbb{D}[r_j](q_j) \middle| r^* \text{ and } r \text{ are distinguishable}\right] \geq \frac{1}{3}$$

In an effective round, if the recommendation is incorrect, it will be excluded with a probability of at least $\frac{1}{3}$. We start by proving that with a high probability, every good round is effective, i.e., the following inequality holds:

$$\Pr\left[\exists j \in [\beta], \text{round } j \text{ is good but not effective}\right] \leq \beta|\mathbb{Q}|\exp(-\frac{1}{18}n) \tag{5}$$

Suppose round j is a good round, and r_j, r^* are distinguishable, let $P = \{p_1, \dots, p_n\}$ be the set of samples. For a question $q \in \mathbb{Q}$, let $g(q)$ be the probability for a random program to differ with r_j on q, i.e., $g(q) = \Pr_{x \sim \varphi|_{C_j}}[\mathbb{D}[x](q) \neq \mathbb{D}[r_j](q)]$. Since r^* is subject to $\varphi|_{C_j}$, for any $q \in \mathbb{Q}$, we have:

$$\Pr\left[\mathbb{D}[r^*](q) \neq \mathbb{D}[r_j](q) \middle| r^* \text{ and } r \text{ are distinguishable}\right] \geq g(q)$$

Let $h(q)$ be the number of samples which differs with r_j on q. Then, by Chernoff bound:

$$\forall \epsilon' > 0, q \in \mathbb{Q}, \Pr\left[\frac{h(q)}{n} - g(q) > \epsilon'\right] \leq \exp(-2n\epsilon'^2)$$

Take a union bound on all possible questions, we obtain:

$$\forall \epsilon' > 0, \Pr\left[\forall q \in \mathbb{Q}, \frac{h(q)}{n} - g(q) > \epsilon'\right] \leq |\mathbb{Q}|\exp(-2n\epsilon'^2)$$

$$\Longrightarrow \forall \epsilon' > 0, \Pr\left[\frac{h(q_j)}{n} - g(q_j) > \epsilon'\right] \leq |\mathbb{Q}|\exp(-2n\epsilon'^2)$$

Take $\epsilon' = \frac{1}{6}$. Since $h(q_j) \geq \frac{1}{2}n$ (guaranteed by the definition of good rounds), we have:

$$\Pr\left[g(q_j) < \frac{1}{3} \middle| \text{round } j \text{ is good}\right] \leq |\mathbb{Q}|\exp(-\frac{1}{18}n)$$

$$\Longrightarrow \Pr\left[\text{round } j \text{ is good but not effective}\right] \leq |\mathbb{Q}|\exp(-\frac{1}{18}n)$$

Take a union bound on all possible rounds, we obtain Inequality 5.

Now we bound $e_2(EpsSy, \varphi)$ with the help of Inequality 5. Let Δ be the event that every good round is effective, let ξ_j be the event that *EpsSy* returns a wrong recommendation in the jth round though condition two (Line 16 in Algorithm 2). Since ξ_j happens only when a wrong recommendation survives for f_ϵ good rounds, $\Pr[x_j|\Delta]$ is no larger than $\left(\frac{2}{3}\right)^{f_\epsilon}$. Take union bound on all possible rounds, we obtain:

$$\Pr\left[\exists j \in [\beta], \xi_j \middle| \Delta\right] \leq \beta\left(\frac{2}{3}\right)^{f_\epsilon}$$

Therefore, we can bound $e_2(EpsSy, \varphi)$ by n and f_ϵ:

$$\begin{aligned} e_2(EpsSy, \varphi) &\leq \Pr[\neg\Delta] + \Pr\left[\exists j \in [\beta], \xi_j \middle| \Delta\right] \\ &\leq \beta|\mathbb{Q}|\exp(-\frac{1}{18}n) + \beta\left(\frac{2}{3}\right)^{f_\epsilon} \end{aligned}$$

The target theorem can be obtained by setting n and f_ϵ to proper values on which both e_1 and e_2 are bounded:

$$\forall n > \max\left(18\ln\left(\frac{2\beta|\mathbb{Q}|}{\epsilon}\right), \frac{16\ln 2}{\epsilon^2} + \frac{8}{\epsilon^2}\ln\left(\frac{1}{\epsilon}\right)\right),$$

$$\forall f_\epsilon > \log_{3/2}\left(\frac{2\beta}{\epsilon}\right), e(EpsSy, p) \leq \epsilon$$

□

Theorem B.5 (Theorem 5.7). *For a program domain $\mathbb{P}$, an example sequence C and a distribution φ defined by a PCFG, let G be an acyclic VSA that represents $\mathbb{P}|_C$, then* Sample(S) *is subject to the conditional distribution $\varphi|_C$, where S is the start symbol of V.*

Proof. First, we prove that function GetPr(s) is well behaved, i.e., it returns the sum of probabilities of all subprograms which are expanded from s. Formally, for each non-terminal symbol s in G, let $SP(s)$ be the set of subprograms expanded from s; for each subprogram sp, let $w(sp)$ be the probability of sp given by the PCFG. We firstly show that GetPr(s) always returns $\sum_{sp \in SP(s)} w(sp)$.

We prove it by induction on the acyclic structure of G. By the definition of VSA, there are three possible forms of the rule expanding from non-terminal symbol s.

Form $s := p_1|\dots|p_k$ is the initial case of the induction. At this time, $SP(s) = \{p_1, \dots, p_k\}$ and $w(p_i) = \gamma(\sigma(s, p_i))$. Therefore $\textsc{GetPr}(s) = \sum_{i=1}^{k} \gamma(\sigma(s, p_i)) = \sum_{sp \in SP(s)} w(sp)$.

For form $s := s_1|\dots|s_k$, there is a bijection between $SP(s)$ and $\cup_{i=1}^{k} SP(s_i)$. For subprogram sp in $SP(s)$, let sp' be the corresponding subprogram which is expanded from s_i, then $w(sp)$ is equal to $w(sp') \times \gamma(\sigma(s, s_i))$. By these facts, we have:

$$\sum_{sp \in SP(s)} w(sp) = \sum_{i=1}^{k} \left(\sum_{sp' \in SP(s_i)} w(sp') \right) \times \gamma(\sigma(s, s_i))$$

By the induction hypothesis, $\sum_{sp' \in SP(s_i)} w(sp')$ is equal to $\textsc{GetPr}(s_i)$. Therefore:

$$\sum_{sp \in SP(s)} w(sp) = \sum_{i=1}^{k} \textsc{GetPr}(s_i) \times \gamma(\sigma(s, s_i)) = \textsc{GetPr}(s)$$

For form $s := F(s_1, \dots, s_k)$, there is also a bijection between $SP(s)$ and $SP(s_1) \times SP(s_2) \times \dots \times SP(s_k)$. For subprogram sp in $SP(s)$, let $sp_1, \dots, sp_k$ be the corresponding subprograms, then $w(sp)$ is equal to $\gamma(\sigma(s, F)) \prod_{i=1}^{k} w(sp_i)$. By these facts, we have:

$$\begin{aligned} \sum_{sp \in SP(s)} w(sp) &= \gamma(\sigma(s, F)) \left(\sum_{sp_1 \in SP(s_1)} \cdots \sum_{sp_k \in SP(s_k)} \prod_{i=1}^{k} w(sp_i) \right) \\ &= \gamma(\sigma(s, F)) \prod_{i=1}^{k} \left(\sum_{sp_i \in SP(s_i)} w(sp_i) \right) \end{aligned}$$

By the induction hypothesis, $\sum_{sp_i \in SP(s_i)} w(sp_i)$ is equal to $\textsc{GetPr}(s_i)$. Therefore:

$$\sum_{sp \in SP(s)} w(sp) = \gamma(\sigma(s, F)) \prod_{i=1}^{k} \textsc{GetPr}(s_i) = \textsc{GetPr}(s)$$

So far, we have proven that $\textsc{GetPr}$ is well behaved. To prove Theorem B.5, we dicsuss a stronger property instead: for each subprogram sp which is expanded from symbol s in G, the probability for $\textsc{Sample}(s)$ to return sp is proportional to $w(sp)$. Obviously, the target theorem is the special case when $s = S$.

We prove this property by induction on the acyclic structure of G, too. For convenience, we use $\phi_s(sp)$ to denote the probability of choosing sp from symbol s.

For form $s := p_1|\dots|p_k$, according to the definition of $\textsc{Sample}$, $\textsc{Sample}(s)$ picks p_i with the probability proportional to $w(p_i)$. This forms the initial case of the induction.

For form $s := s_1|\dots|s_k$, let sp_1, sp_2 be any two subprograms expanded from non-terminal symbol s, there are two possible cases:

- Both sp_1, sp_2 correspond to the same sub-symbol s_i. The induction hypothesis shows $\phi_s(sp_1) : \phi_s(sp_2) = w(sp_1) : w(sp_2)$ directly.
- sp_1 and sp_2 correspond to two different sub-symbols s_i and s_j. Then by the induction hypothesis:

  $$\phi_{s_i}(sp_1) = w(sp_1)/\gamma(\sigma(s, s_i))\textsc{GetPr}(s_i)$$
  $$\phi_{s_j}(sp_2) = w(sp_2)/\gamma(\sigma(s, s_j))\textsc{GetPr}(s_j)$$

  Since the probability for $\textsc{Sample}(s)$ to choose s_i is proportional to $\gamma(\sigma(s, s_i))\textsc{GetPr}(s_i)$, we obtain $\phi_s(sp_1) : \phi_s(sp_2) = w(sp_1) : w(sp_2)$.

For form $s := F(s_1, \dots, s_k)$, let $x = F(x_1, \dots, x_k), y = F(y_1, \dots, y_k)$ be two subprograms rooted at s. By the induction hypothesis, $\forall i \in [k], \phi_{s_i}(x_i) : \phi_{s_i}(y_i) = w(x_i) : w(x_j)$. Therefore:

$$\frac{\phi_s(x)}{\phi_s(y)} = \frac{\prod_{i=1}^{k} \phi_{s_i}(x_i)}{\prod_{i=1}^{k} \phi_{s_i}(y_i)} = \frac{\prod_{i=1}^{k} w(x_i)}{\prod_{i=1}^{k} w(y_i)} = \frac{w(x)}{w(y)}$$

So far, we show that the induction holds on all three forms. Therefore the target theorem has been proven. □

References

[1] Micah Adler and Brent Heeringa. 2008. Approximating optimal binary decision trees. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 1–9.

[2] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. 1–8.

[3] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. 2017. Scaling Enumerative Program Synthesis via Divide and Conquer. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*. 319–336. https://doi.org/10.1007/978-3-662-54577-5_18

[4] Anonymous. [n. d.]. Supplementary Material. https://github.com/jiry17/IntSy.

[5] Shaon Barman, Rastislav Bodík, Satish Chandra, Emina Torlak, Arka Aloke Bhattacharya, and David Culler. 2015. Toward tool support for interactive synthesis. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2015, Pittsburgh, PA, USA, October 25-30, 2015*. 121–136.

[6] Nikolaj Bjørner and Anh-Dung Phan. 2014. νZ - Maximal Satisfaction with Z3. In *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014*. 1–9. http://www.easychair.org/publications/paper/200953

[7] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νZ - An Optimizing SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. 194–199. https://doi.org/10.1007/978-3-662-46681-0_14

[8] Nader H. Bshouty. 1995. Exact Learning Boolean Functions via the Monotone Theory. *Electronic Colloquium on Computational Complexity (ECCC)* 2, 8 (1995). http://eccc.hpi-web.de/eccc-reports/1995/TR95-008/index.html

[9] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. 1996. Oracles and Queries That Are Sufficient for Exact Learning. *J. Comput. Syst. Sci.* 52, 3 (1996), 421–433. https://doi.org/10.1006/jcss.1996.0032

[10] Venkatesan T Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh Mohania. 2007. Decision trees for entity identification: Approximation algorithms and hardness results. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 53–62.

[11] Venkatesan T Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, and Yogish Sabharwal. 2009. Approximating decision trees with multiway branches. In *International Colloquium on Automata, Languages, and Programming*. Springer, 210–221.

[12] Sanjoy Dasgupta. 2004. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*. 337–344. http://papers.nips.cc/paper/2636-analysis-of-a-greedy-active-learning-strategy

[13] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *TACAS*. Springer, 337–340.

[14] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program synthesis using conflict-driven learning. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. 420–435.

[15] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. [n. d.]. Component-based synthesis of table consolidation and transformation tasks from examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*.

[16] Joel Galenson, Philip Reames, Rastislav Bodik, Bjorn Hartmann, and Koushik Sen. 2014. CodeHint: dynamic and interactive synthesis of code snippets. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 653–663.

[17] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. 317–330.

[18] Sumit Gulwani and Prateek Jain. 2017. Programming by Examples: PL meets ML. In *APLAS*.

[19] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. 2011. Synthesis of loop-free programs. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*. 62–73.

[20] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119.

[21] Anupam Gupta, Viswanath Nagarajan, and R Ravi. 2017. Approximation algorithms for optimal decision trees and adaptive TSP problems. *Mathematics of Operations Research* 42, 3 (2017), 876–896.

[22] Tihomir Gvero, Viktor Kuncak, and Ruzica Piskac. 2011. Interactive Synthesis of Code Snippets. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. 418–423.

[23] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. 215–224.

[24] Susmit Jha and Sanjit A. Seshia. 2017. A theory of formal synthesis via inductive learning. *Acta Inf.* 54, 7 (2017), 693–726.

[25] Mark Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics* 24, 4 (1998), 613–632.

[26] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*. 3363–3372.

[27] Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2014. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. 333–350. https://doi.org/10.1007/978-3-319-09284-3_25

[28] Hyafil Laurent and Ronald L Rivest. 1976. Constructing optimal binary decision trees is NP-complete. *Information processing letters* 5, 1 (1976), 15–17.

[29] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*. 542–553.

[30] Vu Le, Daniel Perelman, Oleksandr Polozov, Mohammad Raza, Abhishek Udupa, and Sumit Gulwani. 2017. Interactive Program Synthesis. *CoRR* abs/1703.03539 (2017).

[31] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. 2018. Accelerating search-based program synthesis using learned probabilistic models. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. 436–449. https://doi.org/10.1145/3192366.3192410

[32] Alan Leung, John Sarracino, and Sorin Lerner. 2015. Interactive parser synthesis by example. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. 565–574.

[33] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. 2014. Symbolic optimization with SMT solvers. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 607–618. https://doi.org/10.1145/2535838.2535857

[34] Fan Long and Martin Rinard. 2016. Automatic patch generation by learning correct code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 298–312. https://doi.org/10.1145/2837614.2837617

[35] Ruben Martins, Jia Chen, Yanju Chen, Yu Feng, and Isil Dillig. 2019. Trinity: An Extensible Synthesis Framework for Data Science. *PVLDB* 12, 12 (2019), 1914–1917.

[36] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin G. Zorn, and Sumit Gulwani. 2015. User Interaction Models for Disambiguation in Programming by Example. In *UIST*. ACM, 291–301.

[37] Sergey Mechtaev, Alberto Griggio, Alessandro Cimatti, and Abhik Roychoudhury. 2018. Symbolic execution with existential second-order constraints. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. 389–399. https://doi.org/10.1145/3236024.3236049

[38] Hong Mei and Lu Zhang. 2018. Can big data bring a breakthrough for software automation? *Sci. China Inf. Sci.* 61, 5 (2018), 056101:1–056101:3. https://doi.org/10.1007/s11432-017-9355-3

[39] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd D. Millstein. 2018. FlashProfile: a framework for synthesizing data profiles. *PACMPL* 2, OOPSLA (2018), 150:1–150:28.

[40] Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati. 2016. Scaling up Superoptimization. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016*. 297–310. https://doi.org/10.1145/2872362.2872387

[41] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*. 107–126.

[42] Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. 2015. An Analysis of Patch Plausibility and Correctness for Generate-and-validate Patch Generation Systems *(ISSTA 2015)*. 24–36.

[43] Greg Schohn and David Cohn. 2000. Less is More: Active Learning with Support Vector Machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. 839–846.

[44] Roberto Sebastiani and Patrick Trentin. 2015. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. 447–454. https://doi.org/10.1007/978-3-319-21690-4_27

[45] Roberto Sebastiani and Patrick Trentin. 2015. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. 335–349. https://doi.org/10.1007/978-3-662-46681-0_27

[46] Armando Solar-Lezama, Rodric M. Rabbah, Rastislav Bodík, and Kemal Ebcioglu. 2005. Programming by sketching for bit-streaming programs. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*. 281–294. https://doi.org/10.1145/1065010.1065045

[47] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. 2006. Combinatorial sketching for finite programs. *ACM SIGOPS Operating Systems Review* 40, 5 (2006), 404.

[48] Armando Solarlezama. 2008. Program synthesis by sketching. *Dissertations & Theses - Gradworks* (2008).

[49] Chenglong Wang, Alvin Cheung, and Rastislav Bodík. 2017. Interactive Query Synthesis from Input-Output Examples. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. 1631–1634.

[50] Yuepeng Wang, Xinyu Wang, and Isil Dillig. 2018. Relational program synthesis. *PACMPL* 2, OOPSLA (2018), 155:1–155:27.

[51] Yingfei Xiong, Jie Wang, Runfa Yan, Jiachen Zhang, Shi Han, Gang Huang, and Lu Zhang. 2017. Precise Condition Synthesis for Program Repair. In *ICSE*. https://doi.org/10.1109/ICSE.2017.45

[52] Kuat Yessenov, Shubham Tulsiani, Aditya Krishna Menon, Robert C. Miller, Sumit Gulwani, Butler W. Lampson, and Adam Kalai. 2013. A colorful approach to text processing by example. In *The 26th Annual ACM Symposium on User Interface Software and Technology, UIST'13, St. Andrews, United Kingdom, October 8-11, 2013*. 495–504.