A PROOFS

This appendix supplies the proofs for the lemmas and theorems in this paper.

A.1 Proof of Thm. 3.3

Theorem A.1 (Soundness of VSA Intersection). *For any set of VSAs $\mathbb{V}$, their intersection represents the set of programs shared among all individual VSAs, or formally, $\mathbb{P}\left(\bigcap_{V\in\mathbb{V}} V\right) = \bigcap_{V\in\mathbb{V}}\mathbb{P}(V)$.*

Proof. For any set of VSAs $\mathbb{V}$, their intersection represents the set of programs shared among all individual VSAs, or formally, $\mathbb{P}\left(\bigcap_{V\in\mathbb{V}} V\right) = \bigcap_{V\in\mathbb{V}}\mathbb{P}(V)$.

To get the target theorem, it is sufficient to prove that the intersection of 2 VSAs is sound. We prove this case by showing a more general result: $\mathbb{P}(\varphi(n, n')) = \mathbb{P}(n) \cap \mathbb{P}(n')$ holds for any node n in the first VSA and any node n' in the second VSA.

Our proof is conducted by induction on the total height of the sub-VSAs rooted at nodes n and n'. If the total height is 0, both n and n' must be leaf nodes. The definition of $\varphi(n, n')$ can only match case 5. By definition, we have

$$\mathbb{P}(\varphi(n, n')) = \mathbb{P}(n) \cap \mathbb{P}(n')$$

Then, for the general case, we perform case analysis on the types of n and n'.

Case 1. If both n and n' are union nodes, then they can be written as $\cup(n_1, \ldots, n_k)$ and $\cup(n'_1, \ldots, n'_{k'})$ respectively. The definition of $\varphi(n, n')$ matches case 1, which is

$$\varphi(n, n') = \cup(\varphi(n_i, n'_j)) \; for \; i \in [1, k] \; and \; j \in [1, k'].$$

The height of nodes $n_1, \ldots, n_k$ and $n'_1, \ldots, n'_{k'}$ are at least 1 lower than node n and n' respectively. So, according to the induction hypothesis, for any $i \in [1, k]$ and $j \in [1, k']$:

$$\mathbb{P}(\varphi(n_i, n'_j)) = \mathbb{P}(n_i) \cap \mathbb{P}(n'_i)$$

From the definition of union nodes, for any nodes $a_1, \ldots, a_m$,

$$\mathbb{P}(\cup(a_1, \ldots, a_m)) = \bigcup_{i=1}^{m} \mathbb{P}(a_i)$$

Combining the equations above, we get

$$\begin{aligned}
\mathbb{P}(\varphi(n, n')) &= \mathbb{P}(\cup(\varphi(n_i, n'_j))) \; for \; i \in [1, k] \; and \; j \in [1, k'] \\
&= \bigcup_{i=1}^{k} \bigcup_{j=1}^{k'} (\mathbb{P}(\varphi(n_i, n'_j))) \\
&= \bigcup_{i=1}^{k} \bigcup_{j=1}^{k'} (\mathbb{P}(n_i) \cap \mathbb{P}(n'_j)) \\
&= \bigcup_{i=1}^{k} \mathbb{P}(n_i) \cap \bigcup_{j=1}^{k'} \mathbb{P}(n'_j) \\
&= \mathbb{P}(\cup(n_1, \ldots, n_k)) \cap \mathbb{P}(\cup(n'_1, \ldots, n'_{k'})) \\
&= \mathbb{P}(n) \cap \mathbb{P}(n')
\end{aligned}$$

Case 2. If only one of n and n' is a union node, the proof is the same as case 1, as any non-union node can be regarded as a union node with one child.

Case 3. If both n and n' are join nodes of the same operator f, then they can be written as $f(n_1, \ldots, n_k)$ and $f(n'_1, \ldots, n'_k)$ respectively.

the definition of $\varphi(n, n')$ matches case 3, which is

$$\mathbb{P}(\varphi(n_i, n'_i)) = f(\varphi(n_i, n'_i))\ for\ i \in [1, k]$$

The height of nodes $n_1, \ldots, n_k$ and $n'_1, \ldots, n'_k$ are at least 1 lower than node n and n' respectively. So, according to the induction hypothesis, for any $i \in [1, k]$:

$$\mathbb{P}(\varphi(n_i, n'_i)) = \mathbb{P}(n_i) \cap \mathbb{P}(n'_i)$$

From the definition of join nodes, for any nodes $a_1, \ldots, a_m$,

$$\mathbb{P}(f(a_1, \ldots, a_m)) = f(\prod_{i=1}^{m} \mathbb{P}(a_i))$$

Combining the equations above, we get

$$\begin{aligned}
\mathbb{P}(\varphi(n, n')) &= \mathbb{P}(f(\varphi(n_i, n'_i)))\ for\ i \in [1, k] \\
&= f(\prod_{i=1}^{k} \mathbb{P}(\varphi(n_i, n'_i))) \\
&= f(\prod_{i=1}^{k} (\mathbb{P}(n_i) \cap \mathbb{P}(n'_i))) \\
&= f(\prod_{i=1}^{k} \mathbb{P}(n_i)) \cap f(\prod_{i=1}^{k} \mathbb{P}(n'_i)) \\
&= \mathbb{P}(n) \cap \mathbb{P}(n')
\end{aligned}$$

Case 4. If both n and n' are join nodes of different operators f and g respectively, the outermost operators of their corresponding programs must be f and g respectively.

As the outermost operators are different,

$$\mathbb{P}(n) \cap \mathbb{P}(n') = \emptyset$$

The definition of $\varphi(n, n')$ can only match case 4, so we have

$$\varphi(n, n') = \emptyset = \mathbb{P}(n) \cap \mathbb{P}(n')$$

Case 5. If one of n and n' is a leaf node, the definition of $\varphi(n, n')$ can only match case 5. From the definition of $\varphi(n, n')$ in case 5, we have

$$\mathbb{P}(\varphi(n, n')) = \mathbb{P}(n) \cap \mathbb{P}(n')$$

□

A.2 Proof of Lem. 3.6

Lemma A.2 (Case Study). *Consider the following grammar that involves a constant c, an input variable x, a commutative and associative operator $\oplus$, and a depth limit of h.*

$$S_i \coloneqq S_{i+1} \oplus S_{i+1} \quad \textbf{for}\ i \in [1, h) \qquad\qquad S_h \coloneqq c \mid x$$

Let $\mathbb{V}$ be a set of VSAs on this grammar, each constructed for a given IO example. Then, the formula below provides a lower-bound for the number of nodes constructed by the top-down algorithm when intersecting $\mathbb{V}$, where $\#jnode(V, i)$ denotes the number of join nodes related to S_i.

$$\sum_{i=1}^{h} \prod_{V \in \mathbb{V}} \#jnode(V, i)$$

If we further assume the VSA nodes distribute uniformly at each level, this lower-bound will become $(2h)^{1-|\mathbb{V}|}\prod_{V\in\mathbb{V}}|V|$*, which is close to the size of the Cartesian product.*

Proof. Consider the following grammar that involves a constant c, an input variable x, a commutative and associative operator $\oplus$, and a depth limit of h.

$$S_i \coloneqq S_{i+1}\oplus S_{i+1}\quad \textbf{for } i\in[1,h) \qquad\qquad S_h \coloneqq c \mid x$$

Let $\mathbb{V}$ be a set of VSAs on this grammar, each constructed for a given IO example. Then, the formula below provides a lower-bound for the number of nodes constructed by the top-down algorithm when intersecting $\mathbb{V}$, where $\#\mathsf{jnode}(V,i)$ denotes the number of join nodes related to S_i.

$$\sum_{i=1}^{h}\prod_{V\in\mathbb{V}}\#\mathsf{jnode}(V,i)$$

If we further assume the VSA nodes distribute uniformly at each level, this lower-bound will become $(2h)^{1-|\mathbb{V}|}\prod_{V\in\mathbb{V}}|V|$, which is close to the size of the Cartesian product.

Because of the way single example VSAs are constructed, they must follow a h-level structure in this lemma. The lower part of each level is join nodes combining nodes at a lower level, and the upper part is union nodes merging join nodes corresponding to the same output.

We prove a stronger version of the first part of lemma 3.6, denoted as lemma 3.6':

When intersecting VSAs with the above structure, the top-down algorithm will construct a node for every tuple of join nodes at the same level.

We use induction on the level the node tuple is on.

On level 1, the top-down algorithm constructs the Cartesian product of all join nodes on level 1 when traversing the root node, so a node for every tuple of join nodes is constructed.

Suppose that a node for every tuple of join nodes on level $n-1$ is constructed.

We consider an arbitrary tuple of join nodes on level n $(n_1,\dots,n_k)$.

We denote the tuple of their union node fathers (or in the case that they do not have a union node father, themselves) $(f_1,\dots,f_k)$, and the fathers of $(f_1,\dots,f_k)$ $(t_1,\dots,t_k)$. $(t_1,\dots,t_k)$ are all join nodes because of the structure of individual VSAs.

f_i may be either the left child or right child of t_i. We construct another tuple of join nodes on level $n-1$ $(t'_1,\dots,t'_k)$ where f_i is always the left child of t'_i.

If f_i is the left child, $t'_i = t_i$. If f_i is the right child, because of the commutativity of $\oplus$, another join node must exist where f_i is the left child. We take this node as t'_i.

The resulting tuple $(t'_1,\dots,t'_k)$ is a tuple of join nodes on level $n-1$, so a node must be constructed in the intersection VSA.

When this node is constructed, its children is also created. In the case of $(t'_1,\dots,t'_k)$, the left child is created from the node tuple $(f_1,\dots,f_k)$.

The node tuple $(f_1,\dots,f_k)$ is a mix of union and join nodes on level n, so the intersection node is a union node whose children is the Cartesian product of the children set of each union node, and a set containing only the node itself for each non-union node. Either way, the set corresponding to node f_i always contain the node n_i, so $(n_1,\dots,n_k)$ is included in the Cartesian product, and a node is constructed for the tuple. Thus lemma 3.6' holds for every level in a VSA.

The number of tuples of join nodes at level i is $\prod_{V\in\mathbb{V}}\#\mathsf{jnode}(V,i)$, so the total number of nodes constructed is at least $\sum_{i=1}^{h}\prod_{V\in\mathbb{V}}\#\mathsf{jnode}(V,i)$.

If we further assume the VSA nodes distribute uniformly at each level, each level would have exactly $|V|/h$ nodes. Because each union node represents program sets that have the same output, each join node on each level may be the child to at most one union node, so at least half of nodes are join nodes at each level. Therefore, each level would have at least $|V|/2h$ join nodes. Replace $\#\mathsf{jnode}(V,i)$ in the first part with $|V|/2h$, and the second part of lemma 3.6 follows.

□

A.3 Proof of Thm. 3.7

Theorem A.3. *Let G be any grammar involving commutative and associative operators, and let $\mathbb{V}$ be a set of VSAs on this grammar, each constructed for a given IO example. Then, the following formula provides a lower-bound for the number of constructed nodes when invoking the top-down algorithm on $\mathbb{V}$.*

$$\sum_{\oplus \in G} \sum_{i>0} \prod_{V \in \mathbb{V}} \#jnode(V, \oplus, i)$$

where $\oplus$ ranges over the commutative and associative operators in G, and $\#jnode(V, \oplus, i)$ denotes the number of join nodes in V reachable from the root by passing only union nodes and exactly i $\oplus$-labeled join nodes.

Proof. Let G be any grammar involving commutative and associative operators, and let $\mathbb{V}$ be a set of VSAs on this grammar, each constructed for a given IO example. Then, the following formula provides a lower-bound for the number of the constructed nodes when invoking the top-down algorithm on $\mathbb{V}$.

$$\sum_{\oplus \in G} \sum_{i>0} \prod_{V \in \mathbb{V}} \#\mathsf{jnode}(V, \oplus, i)$$

where $\oplus$ ranges over the commutative and associative operators in G, and $\#\mathsf{jnode}(V, \oplus, i)$ denotes the number of join nodes in V reachable from the root by passing only union nodes and exactly i $\oplus$-labeled join nodes.

We consider a subset of each V_i in $\mathbb{V}$ for each commutative and associative operator $\oplus$, denoted as $V_{i\oplus}$. The set of $V_{i\oplus}$ is denoted as $\mathbb{V}_{\oplus}$.

$V_{i\oplus}$ is defined as the VSA whose nodes are the root node and nodes in V_i that are only reachable from the root by passing only union nodes and $\oplus$-labeded join nodes, and whose edges are the edges in V_i that connect these nodes.

When we invoke the top-down algorithm on $\mathbb{V}$, the resulting VSA contains all the nodes from the VSA resulting from invoking the algorithm on $\mathbb{V}_{\oplus}$.

Following lemma 3.6, the VSA resulting from invoking the algorithm on $\mathbb{V}_{\oplus}$ contains at least $\sum_{i>0} \prod_{V \in \mathbb{V}} \#\mathsf{jnode}(V, \oplus, i)$ nodes.

For each commutative and associative operator $\oplus$ and each i, the node sets of $V_{i\oplus}$ are disjoint except for the root, so the top-down intersection of each $V_{i\oplus}$ has disjoint node sets except for the root.

Therefore, each operator $\oplus$ contributes at least $\sum_{i>0} \prod_{V \in \mathbb{V}} \#\mathsf{jnode}(V, \oplus, i)$ nodes in the complete top-down intersection.

It follows that at least $\sum_{\oplus \in G} \sum_{i>0} \prod_{V \in \mathbb{V}} \#\mathsf{jnode}(V, \oplus, i)$ nodes are in the complete top-down intersection. □

A.4 Proof of Thm. 4.4

Theorem A.4 (Soundness). *Given a set $\mathbb{V}$ of VSAs, the output of Algorithm 2 is equal to the core intersection $core(\bigcap_{V \in \mathbb{V}} V)$ if no node in $\mathbb{V}$ has both union parents and join parents.*

Proof. Given a set $\mathbb{V}$ of VSAs, the output of Algorithm 2 is equal to the core intersection $\mathrm{core}(\bigcap_{V \in \mathbb{V}} V)$ if no node in $\mathbb{V}$ has both union parents and join parents.

We assume that $\mathrm{core}(\bigcap_{V \in \mathbb{V}} V)$ has height h. We use V_i to denote the set of all nodes in $\mathrm{core}(\bigcap_{V \in \mathbb{V}} V)$ that have height i V_i.

We prove by induction that all nodes in $\mathrm{core}(\bigcap_{V \in \mathbb{V}} V)$ is constructed by the bottom-up algorithm, by proving that the set N in the bottom-up algorithm contains every node in $V_1 \cup \ldots V_i$ before the ith invocation of function buildUp.

For the case of the 1st invocation, each node in V_1 must be a leaf node, representing programs directly. If a program p is represented by some leaf node v in V_1, the program must be represented in a leaf node in each individual VSA. The bottom-up algorithm constructs a node for each different program represented by leaf nodes, so the lemma holds for this case.

Suppose that the lemma holds for V_1 through V_{i-1}. We now prove the lemma for V_i. Each node in V_i is either a join node or a union node.

Consider a join node v in V_i with operator f, constructed from the node tuple $(v^1, \ldots, v^n)$. The children of v are $v_1, \ldots, v_m$. The children of v^i are denoted as $v_1^i, \ldots, v_m^i$ respectively.

Because v is a join node, it must be constructed by intersecting join nodes, and its ith child v_i is the intersection of the node tuple $(v_i^1, \ldots, v_i^n)$. From the induction hypothesis, $v_1, \ldots, v_m$ are all in N, so all the children of $v^1, \ldots, v^n$ are used in some node in N.

Therefore, the node tuple $(v^1, \ldots, v^n)$ is found in the buildUp function, and a node is constructed for the tuple.

Consider a union node v in V_i, constructed from the node tuple $(v^1, \ldots, v^n)$, with children $v_1, \ldots, v_m$.

The intersection is only necessary if node v^i has a join father or is the root of its VSA, which means node v^i has no union father. The program set of a VSA root is the program set of the entire VSA. A join node takes the Cartesian product of the program sets of each child and constructs new programs using the product. These are the only two ways a set of programs need to exist as an individual node.

Union nodes are constructed by taking the intersection of each tuple in the Cartesian product of their respective children set. Therefore, each v_i is constructed by a tuple of nodes $(v_i^1, \ldots, v_i^n)$. For a fixed i, v_i^j is a child of v^j if v^j is a union node, and v^j itself if it's a join node. From the induction hypothesis, each node in $v_1, \ldots, v_m$ has already been constructed by the bottom-up algorithm. Therefore, the intersections of node tuples $(v_i^1, \ldots, v_i^n)$ have all been constructed in N.

We take $\varphi(v_1^1, \ldots, v_1^n)$. If v_1^j is a child of v^j, v^j must be a union father of it, so it is inserted into U_j. If $v_1^j = v^j$, because v^j does not have a union father, v^j itself is inserted into U_j.

Therefore, we construct every node in $\text{core}(\bigcap_{V \in \mathbb{V}} V)$.

Note that every node is constructed from a node tuple in the Cartesian product of the node sets of each VSA. Thus, every node we construct is included in $\bigcap_{V \in \mathbb{V}} V$. Take the core set of both node sets, and our core set is a subset of $\text{core}(\bigcap_{V \in \mathbb{V}} V)$.

Therefore, we construct the same VSA as $\text{core}(\bigcap_{V \in \mathbb{V}} V)$.

□

A.5 Proof of Lem. 4.5

Lemma A.5 (Case Study). *Consider the grammar in Lem. 3.6, repeated as follows, where c is a constant, x is the input variable, $\oplus$ is a commutative and associative operator, and h is a depth limit.*

$$S_i \coloneqq S_{i+1} \oplus S_{i+1} \quad \textbf{for } i \in [1, h) \qquad\qquad S_h \coloneqq c \mid x$$

Let $\mathbb{V}$ be a set of VSAs on this grammar, each constructed from a given IO example. Then, Algorithm 2 will construct at most $h \cdot size^2(p^*)$ nodes when intersecting the VSAs in $\mathbb{V}$.

Proof. The individual VSAs follow a h-level structure. The worst case is when every program constructed from the grammar is a valid program in the intersection VSA.

Assume that level i has n_i union nodes. For level $i+1$, the bottom-up algorithm first constructs a join node for each pair of union nodes in level i, totaling n_i^2 nodes. Then, because of the commutativity and associativity of operator $\oplus$, there are at most $2^i + 1$ different outputs, so only $2^i + 1$ union nodes are constructed.

Therefore, for level i, there are at most $(2^{h-2}+1)^2$ join nodes and $2^{h-1}+1$ union nodes. The size of the largest program p^* is 2^h-1. Each level has more nodes than the level below it, so the largest level is level h with $(2^{h-2}+1)^2+2^{h-1}+1$ nodes, and each level has at most $(2^{h-2}+1)^2+2^{h-1}+1$ nodes, which is less than $\text{size}^2(p^*)$ for all $h \geq 2$. Therefore, the entire VSA consisting of h layers has at most $h \cdot \text{size}^2(p^*)$ nodes.

□

A.6 Proof of Thm. 4.6

Theorem A.6 (Worst-Case Efficiency). *The following inequality always holds for any finite program space P and any example set E.*

$$cost_{Ours}(\mathbb{V}) \leq 2cost_{OE}(P, E)$$

where

- *$\mathbb{V}$ is the set of single-example VSAs constructed over program space P and example set E.*
- *$cost_{Ours}(\mathbb{V})$ denotes the number of VSA nodes constructed by our bottom-up algorithm when intersecting the VSAs in $\mathbb{V}$.*
- *$cost_{OE}(P, E)$ denotes the number of programs constructed by OE after traversing the whole program space P with the example set E.*

Proof. We consider the process in which OE constructs programs. OE maintains a set N that is the set of enumerated programs modulo observational equivalence. When a program is inserted into N, if a program with the same output behavior already exists in N, the program is discarded. For each height h and each operator f, OE enumerates all programs of height h by applying the operator f on programs in N. Then, the enumerated programs are inserted into N.

We define $N(p)$ as the program with identical output to p in n, or p itself if no program with identical output exists in N.

When our bottom-up algorithm constructs a join node of height h and operator f, it must correspond to at least one program $p = f(p_1, \ldots, p_n)$. The lower nodes considered in construction each have different output behavior, so p is unique for each node constructed. Because OE traverses the entire program space, it must construct another program $p' = f(N(p_1), \ldots, N(p_n))$ when enumerating programs with height h.

So, each join node corresponds to a unique program p' constructed by OE, and the number of join nodes is no larger than the number of programs constructed by OE.

Because union nodes combine join nodes with the same output behavior, one join node can have at most one union father, and the number of union nodes is no larger than the number of join nodes.

So, the total number of nodes is no larger than twice the number of join nodes, which is no larger than the number of programs constructed by OE. The theorem follows. □

B IMPLEMENTATION DETAILS

This appendix supplies the implementation details of some algorithms in this paper.

B.1 Details of Algorithm 2

Here we describe the implementation details of the construction of sets L_i, J_i and U_i.

For sets L_i (Line 6) ,before the loop (Lines 5-8) we preprocess each VSA and construct a map from each program to the set of leaf nodes comprising it. Then, the sets L_i can be taken out directly from these maps.

For sets J_i (Line 15) , for each operator f and each VSA, we maintain the set J_i as global variables and incrementally grow them as more tuples are inserted into the set N. Specifically, for each f-join node in VSA V_i, we maintain an auxiliary value denoting the number of its children that are not used in N – this value is initialized as the arity of f. Then, each time when a new tuple $\overline{n_i}$ is added to N, we visit each f-join parent of n_i, decrease its auxiliary value by 1, and add it to the corresponding J_i if the value becomes 0.

For sets U_i (Line 21), we directly enumerate the union parents of n_i. This process does not cost much time because U_i is typically small.

B.2 Details of Algorithm 3

Here we describe the details of the construction of trie for each node set N_{pre}.

Instead of constructing a new trie for each N_{pre}, we maintain the whole node set N as a trie and attach each trie node with a timestamp representing the index of the iteration (Line 9, algorithm 2) when the node is constructed. When each node is inserted into N (Line 4, algorithm 2), we set its timestamp to the index of the current iteration. During search on the trie, we ignore nodes whose timestamps are equal to the index of the current iteration, as these nodes are constructed in the current iteration and are not in the set N_{pre}.