A Appendix

In this section, we complete the proofs of the lemmas and the theorems in our paper.

A.1 Proofs for Section 3.4

Theorem A.1 (Theorem 3.6)**.** *Given preorder $R = \{(op_i, k_i)\}$, define function $N_R(S)$ as the following, where $range(k, S)$ is the range of k on S, i.e., $\max_{a\in S}(k\ a) - \min_{a\in S}(k\ a)$, and $\max(S)$ returns the largest element in S with default value 1.*

$$N_R(S) := \left(\prod_i range(k_i, S)\right) \Big/ \max_i \left(range(k_i, S) \;\middle|\; op_i \in \{\leq\}\right)$$

- *For any set S, $|thin\ R\ S| \leq N_R(S)$.*
- *There is an implementation of $(thin\ R)$ with time complexity $O(N_R(S) size(R) + T_R(S))$, where S is the input set, $size(R)$ is the number of comparisons in R, $T_R(s)$ is the time complexity of evaluating all key functions in R for all elements in S.*

Proof. Let $K_{\leq}$ be the set of key functions in R with operator $\leq$, and let k^* be the key function in $K_{\leq}$ with the largest range on S, i.e., $\arg\max range(k, S), k \in K_{\leq}$. Especially, when $K_{\leq}$ is empty, k^* is defined as the constant function $\lambda x.0$.

Let $K = \{k_1, \dots, k_m\}$ be the set of key functions in R excluding k^*. According to the definition of $N_R(S)$, we have the following equality.

$$N_R(S) = \prod_{i=1}^{m} range(k_i, S)$$

We start with the first claim. Define feature function f_k as $k_1 \triangle \dots \triangle k_m$. Then $N_R(S)$ is the size of f_k's range. By the definition of the keyword preorder, we have the following formula:

$$f_k\ a = f_k\ b \to (aRb \leftrightarrow k^*\ a \leq k^*\ b)$$

In other words, for elements where the outputs of the feature function are the same, their order in R is a total order. Therefore, the number of maximal values in S is no more than the size of the feature function's range, i.e., $N_R(S)$.

Then, for the second claim, Algorithm 3 shows an implementation of *thin*. The time complexity of the first loop (Lines 6-10) is $O(T_R(S))$ and the time complexity of the second loop (Lines 11-20) is $O(N_R(S) size(R))$. Therefore, the overall time complexity of Algorithm 3 is $O(N_R(S) size(R) + T_R(S))$.

The remaining task is to prove the correctness of Algorithm 3. Let $\mathcal{A}_x$ be the algorithm weakened from Algorithm 3 by replacing the loop upper bound in Line 11 from m to x. Besides, let R_x be the keyword preorder $\{(op_i, id)_{i=1}^{x}\} \cup \{(=, id)_{i=x+1}^{m}\}$. Now, consider the following claim.

- After running $\mathcal{A}_x$ on set S, the value of $Val[w]$ is equal to $\arg\max_a k^*\ a, a \in S \wedge wR_x(f_k\ a)$. If there is no such a exist, $Val[w]$ is equal to $\perp$.

If this claim holds, after running $\mathcal{A}_m$, i.e., Algorithm 3 on S, $Val[f_k\ a] = a$ if and only if a is a local maximal in S. Therefore, we get the correctness of Algorithm 3.

Algorithm 3: An implementation of *thin R*.

Input: A set S of elements.
Output: A subset including all maximal values in S.
1 Extract k^* and $f_k = k_1\triangle \dots \triangle k_m$ from R;
2 $op_i \leftarrow$ the operator corresponding to k_i;
3 $[mi_i, ma_i] \leftarrow$ the range of k_i on S;
4 $\mathbb{W} \leftarrow [mi_1, ma_1] \times [mi_2, ma_2] \times \dots \times [mi_m, ma_m]$;
5 $\forall w \in \mathbb{W}, Val[w] \leftarrow \perp$;
6 **foreach** $a \in S$ **do**
7   **if** $Val[f_k\ a] = \perp \vee k^*\ (Val[f_k\ a]) \leq k^*\ a$ **then**
8     $Vak[f_k\ a] \leftarrow a$;
9   **end**
10 **end**
11 **foreach** $i \in [1, m]$ **do**
12   **if** $op_i \in \{=\}$ **then continue**;
13   **foreach** $w \in \mathbb{W}$ *in the decreasing order of $w.i$* **do**
14   **if** $w.i = mi_i \vee Val[w] = \perp$ **then continue**;
15     $w' \leftarrow w$; $w'.i \leftarrow w.i - 1$;
16   **if** $Val[w'] = \perp \vee k^*\ Val[w'] \leq k^*\ Val[w]$ **then**
17       $Val[w'] \leftarrow Val[w]$;
18   **end**
19   **end**
20 **end**
21 **return** $\{a \mid a \in S \wedge Val[f_k\ a] = a\}$;

To prove this claim, we make an induction on the value of m. First, when m is equal to 0, this claim holds because only the element with the largest output of k^* is retained while initializing *Val* (Lines 6-10).

Then, for any $x \in [1, m]$, assume that the claim holds for $\mathcal{A}_{x-1}$. When op_x is equal to $=$, the correctness of $\mathcal{A}_{x-1}$ directly implies the correctness of $\mathcal{A}_x$. Therefore, we consider only the case where $op_x \in \{\leq\}$ below.

Let Val' be the value of Val after running $\mathcal{A}_{x-1}$, and let Val be the value of Val after running $\mathcal{A}_x$. For any $w \in W$ and $i \in [mi_x, ma_x]$, let w_i be the feature that $\forall j \neq x, w_i.j = w.j$ and $w_i.x = i$. According to Lines 11-20 in Algorithm 3, $Val[w]$ is equal to the element with the largest output of k^* among $Val'[w_{w.i}], Val'[w_{w.i+1}], \dots, Val'[w_{ma_x}]$. (For simplicity, we define $k^*\ \perp$ as $-\infty$).

Assume that the claim does not hold for $\mathcal{A}_x$. Then, there exists $w \in \mathbb{W}$ and $a \in S$ satisfying the following formula.

$$\begin{aligned} & k^*\ Val[w] < k^*\ a \wedge wR_x(f_k\ a) \\ \implies & k^*\ Val'\,[w_{k_x\ a}] < k^*\ a \wedge w_{k_x\ a}R_{x-1}(f_k\ a) \end{aligned}$$

This fact contradicts with the inductive hypothesis and thus the induction holds, i.e., the claim holds for all $\mathcal{A}_x$. □

A.2 Proofs for Section 4.1

In Section 4.1, for simplicity, we assume that each transition, i.e., each element in the output of ϕ, involves at most one search state. Therefore, in this section, we first extend Section

4.2 to general relational hylomorphisms and then prove the generalized lemmas.

Let $(h = [\![\phi, \psi]\!]_{\mathsf{F}}, o)$ be the input program. We first generalize the concept of $\twoheadrightarrow_s$. When there is a transition involving multiple states, a partial solution may be constructed from multiple partial solutions. For any state s, let $s_1, \dots, s_n$ be a series of states in $T_h\ s$, and $p_1, \dots, p_n$ be a series of partial solutions where $\forall i \in [1, n], p_i \in h\ s_i$. We use $(p_1, \dots, p_n) \twoheadrightarrow_s p$ to denote that p can be constructed from partial solutions $p_1, \dots, p_n$, i.e., there exists an invocation of ϕ in $h\ s$ where the input includes $p_1, \dots, p_n$ and the output is p.

Using the generalized notation $\twoheadrightarrow$, we generalize Lemma 4.1 to general relational hylomorphisms as the following.

Lemma A.2 (Lemma 4.1). *Given instance i and program $(h = [\![\phi, \psi]\!]_{\mathsf{F}}, o)$, $(rg((thin\ ?R) \circ cup \circ \mathsf{P}\phi, \psi)_{\mathsf{F}}, o) \sim_i ([\![\phi, \psi]\!]_{\mathsf{F}}, o)$ for keyword preorder $?R$ if $?R$ satisfies that (1) $(\leq, o) \in ?R$, and (2) for any state $s \in S_h\ i$, any sequence of states $s_1, \dots, s_n \in (T_h\ s)$, and any two sequences of partial solutions $\overline{p_1} = (p_{1,1}, \dots, p_{1,n})$ and $\overline{p_2} = (p_{2,1}, \dots, p_{2,n})$ where $p_{1,i}, p_{2,i} \in h\ s_i$, $\wedge_{i=1}^n p_{1,i}?Rp_{2,i}$ implies that partial solutions generated from $\overline{p_1}$ are dominated by partial solutions generated from $\overline{p_2}$ in the sense of $?R$, i.e.,*

$$\bigwedge_{i=1}^{n} p_{1,i}?Rp_{2,i} \rightarrow \forall p_1', \left(\overline{p_1} \twoheadrightarrow_s p_1' \rightarrow \exists p_2', \overline{p_2} \twoheadrightarrow_s p_2' \wedge p_1'?Rp_2'\right) \tag{10}$$

Proof. For simplicity, we use r to denote $rg((thin\ ?R) \circ cup \circ \mathsf{P}\phi, \psi)_{\mathsf{F}}$. Because ψ in h is also used in r, the search trees generated by r and h on instance i are exactly the same.

For simplicity, we use $S_1 \sqsupseteq_R S_2$ to denote that elements in S_1 dominates elements in S_2 in the sense of preorder R, i.e., $\forall a \in S_2, \exists b \in S_1, aRb$. By the definition of *thin*, for any preorder R and any set S, $thin\ R\ S \sqsupseteq_R S$ always holds.

Let us consider the following claim.

- For any state s in $S_h\ i$, $r\ s \subseteq h\ s \wedge r\ s \sqsupseteq_{?R} h\ s$.

Let p_o be any solution with the largest objective value in $h\ i$. If this claim holds, there must be a solution p^* in $r\ i$ such that $p_o?Rp^*$. By the precondition that $(\leq, o) \in ?R$, $o\ p^* \geq o\ p_o$. Because $p^* \in r\ i \subseteq h\ i$, we have $o\ p^* = o\ p_o$. Therefore, at least one solution with the largest objective value are retained in $r\ i$, which implies that $(r, o) \sim_i (h, o)$.

We prove this claim by structural induction on the search tree. First, $r\ s \subseteq h\ s$ can be obtained by the definition of rg and $[\![\phi, \psi]\!]_{\mathsf{F}}$. Let us unfold the definition of h and r.

$$\begin{aligned} h &= cup \circ \mathsf{P}\phi \circ cup \circ \mathsf{P}(car[\mathsf{F}] \circ \mathsf{F}h) \circ \psi \\ r &= thin?R \circ cup \circ \mathsf{P}\phi \circ cup \circ \mathsf{P}(car[\mathsf{F}] \circ \mathsf{F}r) \circ \psi \end{aligned}$$

Starting from the inductive hypothesis, we have the following derivation.

$$\begin{aligned} &\forall s' \in T_h\ s, r\ s' \subseteq h\ s' \\ \Longrightarrow\ &\forall t \in \psi\ s, (car[\mathsf{F}] \circ \mathsf{F}r)\ t \subseteq (car[\mathsf{F}] \circ \mathsf{F}h)\ t \\ \Longrightarrow\ &(cup \circ \mathsf{P}(car[\mathsf{F}] \circ \mathsf{F}r) \circ \psi)\ s \subseteq \\ &\qquad (cup \circ \mathsf{P}(car[\mathsf{F}] \circ \mathsf{F}h) \circ \psi)\ s \\ \Longrightarrow\ &r\ s \subseteq h\ s \end{aligned}$$

By the induction, we prove that $\forall s \in S_h\ i, r\ s \subseteq h\ s$.

The remaining task is to prove $\forall s \in S_h\ i, r\ s \sqsupseteq_{?R} h\ s$. For any state s, let P_s be the set of partial solutions constructed in $r\ s$ before applying *thin* $?R$. Let us consider another claim.

- For any state s in $S_h\ i$, $P_s \sqsupseteq_{?R} h\ s$.

If the second claim holds, we prove the first claim by

$$r\ s = thin\ ?R\ P_s \sqsupseteq_{?R} P_s \sqsupseteq_{?R} h\ s$$

Therefore, we need only to prove the second claim via the inductive hypothesis. Suppose this claim does not hold for state s.

$$P_s \not\sqsupseteq_{?R} h\ s \implies \exists p \in h\ s, \forall p' \in P_s, \neg p?Rp' \tag{11}$$

Suppose partial solution p is constructed from partial solutions $p_1, \dots, p_n$ where p_i is taken from state s_i. By the inductive hypothesis, for each $i \in [1, n]$, there exists $p_i' \in r\ s_i$ such that $p_i?Rp_i'$. Let $\overline{p} = (p_1, \dots, p_n)$ and $\overline{p'} = (p_1', \dots, p_n')$.

$$\begin{aligned} \bigwedge_{i=1}^{n} p_i?Rp_i' &\Longrightarrow \forall p_1', \left(\overline{p} \twoheadrightarrow_s p_1' \rightarrow \exists p_2', \overline{p'} \twoheadrightarrow_s p_2' \wedge p_1'?Rp_2'\right) \\ &\Longrightarrow \exists p_2', \overline{p'} \twoheadrightarrow_s p_2' \wedge p?Rp_2' \\ &\Longrightarrow \exists p_2' \in P_s, p?Rp_2' \end{aligned} \tag{12}$$

Formula 12 contradicts with Formula 11. Therefore, we prove the second claim, and thus the induction holds. □

After the generalization, given a preorder R and an instance i, a set of counterexamples $CE(R, i)$ for R can also be extracted from Formula 10. A counter example is a sequence of pairs of partial solutions $((p_{1,j}, p_{2,j})_{j=1}^n)$, representing that $\neg p_{1,j}Rp_{2,j}$ is expected to hold for at least one $j \in [1, n]$.

The following is the generalized version of Lemma 4.2.

Lemma A.3 (Lemma 4.2). *Given instance i, for any keyword preorders $R_1 \subseteq R_2$, the following formula is always satisfied.*

$$\forall e \in CE(R_1, i), e \notin CE(R_2, i) \leftrightarrow \exists (p_1, p_2) \in e, \neg p_1 (R_2/R_1) p_2$$

where R_2/R_1 represents the keyword preorder formed by the new comparisons in R_2 compared with R_1.

Proof. We start with the $\leftarrow$ direction. Suppose there is an example e in $CE(R_1, i)$ satisfying $\exists (p_1, p_2) \in e, \neg p_1 (R_2/R_1) p_2$. Because the comparisons in R_2/R_1 is a subset of R_2, this precondition implies that $\exists (p_1, p_2) \in e, \neg p_1 R_2 p_2$. By Formula 10, e cannot be a counter example for R_2, i.e., $e \notin CE(R_2, i)$.

For the $\rightarrow$ direction, suppose there is an example e in $CE(R_1, i)$ such that $\forall (p_1, p_2) \in e, p_1 (R_2/R_1) p_2$.

Let $(p_{1,1}, p_{2,1}), \dots, (p_{1,n}, p_{2,n})$ be all pairs in example e, let $\overline{p_1}$ and $\overline{p_2}$ be the sequences of $p_{1,j}$ and $p_{2,j}$ respectively. By the definition of CE, we have (1) $\forall (p_1, p_2) \in e, p_1 R_1 p_2$, and (2) the following formula.

$$\begin{aligned} &\exists p_1', \overline{p_1} \twoheadrightarrow_s p_1' \land \forall p_2', \left(\overline{p_2} \twoheadrightarrow_s p_2' \to \neg p_1' R_1 p_2'\right) \\ \Longrightarrow\ &\exists p_1', \overline{p_1} \twoheadrightarrow_s p_1' \land \forall p_2', \left(\overline{p_2} \twoheadrightarrow_s p_2' \to \neg p_1' R_2 p_2'\right) \end{aligned} \tag{13}$$

By the definition of keyword preorders, we have the following derivation.

$$\begin{aligned} &\forall (p_1, p_2) \in e, p_1 R_1 p_2 \land \forall (p_1, p_2) \in e, p_1 (R_2/R_1) p_2 \\ \Longrightarrow\ &\forall (p_1, p_2) \in e, \forall (op, k) \in R_2, (k\ p_1) op (k\ p_2) \\ \Longrightarrow\ &\forall (p_1, p_2) \in e, p_1 R_2 p_2 \end{aligned} \tag{14}$$

Combining Formula 14 with 13, we know example e is in $CE(R_2, i)$, and the other direction of this lemma is proved. □

A.3 Proofs for Section 4.2

Lemma A.4 (Lemma 4.3). *Given instance i and program $prog_2$ in Form 3, let $prog_2'$ be the program constructed in Step 2. $prog_2 \sim_i prog_2'$ if for any query q and any constructor m, Formula 6 and Formula 7 are satisfied respectively.*

$$\forall e \in RE(q, i), q\ e = ?q[q]\ (\mathsf{F}[q] ?f_p\ e) \tag{15}$$

$$\forall e \in RE(m, i), ?f_p\ (m\ e) = ?c[m]\ (\mathsf{F}[m] ?f_p\ e) \tag{16}$$

Proof. Recall the form of $prog$ and $prog_2'$ as the following.

$$\begin{aligned} prog_2 &= (rg((thin\ ?R) \circ cup \circ \mathsf{P}\phi, \psi)_\mathsf{F}, o) \\ prog_2' &= (rg((thin\ R') \circ cup \circ \mathsf{P}\phi', \psi)_\mathsf{F}, ?q[o]) \end{aligned}$$

Comparing $prog_2'$ with $prog_2$, there are several expression-level differences: (1) all key functions in $?R$ are replaced with the corresponding $?q$, (2) the objective function is replaced with $?q[o]$, (3) all solution-related functions in ϕ are replaced with the corresponding $?q$ and $?c$.

Let e_1, e_2 be the small-step executions of $prog_2$ and $prog_2'$ on instance i, and let $e[k]$ be the kth program in execution e. Let us consider the following claim.

- For any k, $e_1[k]$ will be exactly the same with $e_2[k]$ after (1) replacing all solution-related functions with the corresponding $?q$ and $?c$, and (2) replacing all solutions with the corresponding outputs of $?f_p$.

If this claim holds, the last programs in e_1 and e_2 must be exactly the same because they are the outputs of $prog_2$ and $prog_2'$ and include neither functions nor solutions. Therefore, this claim implies that $prog_2 \sim_i prog_2'$.

We prove this claim by induction on the number of steps. When $k = 0$, the claim directly holds because there is no solution constructed and the correspondence of functions is guaranteed by the construction of $prog_2$.

Then for any $k > 0$, consider the kth evaluation rule applied to e_1 and e_2. By the inductive hypothesis, these two evaluation rules must be the same.

- If this evaluation rule relates to partial solutions, it must be the evaluation of a solution-related function. By the inductive hypothesis, (1) the scalar values in both inputs are exactly the same, and (2) the partial solutions used in e_2 are equal to the outputs of $?f_p$ on the partial solutions used in e_1. At this time, the examples used in the synthesis task ensure that the evaluation result is still corresponding.
- If this evaluation rule does not relate to partial solutions, by the inductive hypothesis, the evaluation in e_1 and e_2 must be exactly the same.

Therefore, the induction holds, and thus the claim holds. □

A.4 Proofs for Section 4.3

Lemma A.5 (Lemma 4.4). *Given instance i and program (r, o), where r is a recursive generator, $(r^{?f_m}, o) \sim_i (r, o)$ if for any states $s_1, s_2 \in (S_r\ i)$, $(?f_m\ s_1 = ?f_m\ s_2) \to (r\ s_1 = r\ s_2)$.*

Proof. Consider the following claim.

- Each time when $r^{?f_m}\ s$ returns, (1) the result is equal to $r\ s$, and (2) for any state $s' \in S_r\ i$, there is result recorded with keyword $?f_m\ s'$ implies that the results is $r\ s'$.

If the claim holds, the lemma is obtained by $r^{?f_m}\ i = r\ i$.

Let $r^{?f_m}\ s_1, \dots, r^{?f_m}\ s_n$ be all invocations of $r^{?f_m}$ during $r^{?f_m}\ i$ and they are ordered according to the exit time. We prove the claim by induction on the prefixes of this sequence. For the empty prefix, the claim holds as the memoization space is empty.

Now, consider the kth invocation $r^{?f_m}\ s_k$. There are two cases. In the first case, there has been a corresponding result recorded in the memoization space. At this time, by the inductive hypothesis, this result must be equal to $r\ s_k$, and thus the claims still hold when $r^{?f_m}\ s_k$ returns.

In the second case, there has not been a corresponding result recorded. By the inductive hypothesis, the results of the recursions made by $r^{?f_m}\ s_k$ must be the results of the corresponding recursions made by $r\ s_k$. Therefore, the execution of $r^{?f_m}\ s_k$ must be exactly the same with $r\ s_k$ and thus $r^{?f_m}\ s_k = r\ s_k$. By the examples used to synthesize $?f_m$, we know that for any other state $s \in S_r\ i$, $?f_m\ s = ?f_m\ s_k$ implies that $r\ s = r\ s_k$, i.e., the newly memoized result.

Therefore, the induction holds, and thus the claim holds. □

A.5 Proofs for Section 5.1

Similar to Section 4.1, we generalize Lemma 5.1 to general relational hylomorphisms as the following.

Lemma A.6 (Lemma 5.1). *Given a set of instances I, for any keyword preorders $R_1 \subseteq R_2$ where $\forall i \in I, CE(R_2, i) = \emptyset$, there exists a comparison $(op, k) \in R_2/R_1$ satisfying at least $1/(|R_2| -$*

$|R_1|)$ *portion of examples in* $CE(R_1, I) = \cup_{i \in I} CE(R_1, i)$*, i.e.,*

$$\left|\left\{e \in CE(R_1, I) \;\middle|\; \exists (p_1, p_2) \in e, \neg\Big((k\ p_1) op (k\ p_2)\Big)\right\}\right| \geq \frac{|CE(R_1, I)|}{|R_2| - |R_1|}$$

Proof. Let $(op_1, k_1), \dots, (op_n, k_n)$ be comparisons in R_2/R_1. Define keyword preorders R_x^p as $R_1 \cup \{(op_j, k_j)_{j=1}^x\}$, and define R_x^a as $R_1 \cup \{(op_x, k_x)\}$. By the definition of keyword preorders, this lemma is equivalent to the following formula.

$$\exists x \in [1, n], \left|CE(R_1, I)\middle/CE(R_x^a, I)\right| \geq \frac{|CE(R_1, I)|}{n} \tag{17}$$

We prove Formula 17 in two steps. First, we prove that all $x \in [1, n]$ satisfies the following formula.

$$\left|\Big(CE(R_1, I)/CE(R_x^p, I)\Big)\middle/\Big(CE(R_1, I)\middle/CE(R_{x-1}^p, I)\Big)\right| \leq \left|CE(R_1, I)\middle/CE(R_x^a, I)\right| \tag{18}$$

For any x, let C_x^p be the set in the left-hand side and let C_x^a be the set in the right-hand side. By Lemma A.3, the following derivation holds for any $e \in CE(R_1, I)$.

$$\begin{aligned} e \in C_x^p &\iff \forall (p_1, p_2) \in e, \forall j \in [1, x-1], (k_j\ p_1) op_j (k_j\ p_2) \\ &\qquad \wedge \exists (p_1, p_2) \in e, \exists j \in [1, x], \neg (k_j\ p_1) op_j (k_j\ p_2) \\ &\implies \exists (p_1, p_2) \in e, \neg (k_x\ p_1) op_x (k_x\ p_2) \\ &\iff e \in C_x^a \end{aligned}$$

Therefore, $|C_x^p| \leq |C_x^a|$ and thus Formula 18 is proved.

Then, we prove the following formula.

$$\exists x \in [1, n], |C_x^p| \geq \frac{|CE(R_1, I)|}{n} \tag{19}$$

Because $CE(R_2, I) = \emptyset$, we know $C_1^p \cup C_x^p \cup \dots \cup C_n^p = CE(R_1, I)$. Therefore, $\sum_{i=1}^n |C_i^p| \geq |CE(R_1, I)|$. Let x^* be the index where $|C_x^p|$ is maximized.

$$n\left|C_{x^*}^p\right| \geq \sum_{i=1}^n \left|C_i^p\right| \geq |CE(R_1, I)|$$

Therefore, we prove that Formula 19 holds for $x = x^*$.

As the combination of Formula 18 and Formula 19 directly implies Formula 17, the target lemma is proved. □

Theorem A.7 (Theorem 5.2)**.** *Given program* (h, o)*, a set of instances* I *and a grammar* G *for available comparisons, if there exists a keyword preorder* R *satisfying (1)* $(\leq, o) \in R$*, (2)* $\forall i \in I, CE(R, i) = \emptyset$*, and (3)* R *is constructed by* $(\leq, o)$ *and comparisons in* G*, then MetHyl must terminate and return such a keyword preorder.*

Proof. Let R be any solution satisfying the three conditions. According to Algorithm 2, given a finite set of comparisons and a size limit, function `BestPreorder` always terminates.

We name an invocation of `BestPreorder` good if the comparison space including all comparisons except $(\leq, o)$ used in R and n_c is no smaller than $size(R) - 1$. According to the iteration method used to decide C and n_c, for any t, there will be t good invocations finished within finite time.

Let $(op_1, k_1), \dots, (op_n, k_n)$ be an order of comparisons in $R/\{(\leq, o)\}$ such that for any $x \in [1, n]$, (op_x, k_x) will be a valid comparison for `CandidateComps` in the xth turn if $(op_1, k_1), \dots, (op_{x-1}, k_{x-1})$ are selected in the previous terms. According to Lemma A.6, such an order must exist.

Suppose the error rate of `CandidateComps` is at most c, i.e., the probability for `CandidateComps` to exclude a valid comparison is at most c. For a good invocation of `BestPreorder`, R will be found if for any $x \in [1, n]$, (op_x, k_x) is not falsely excluded in the xth turn by `CandidateComps`. Therefore, the probability for R to be found in a good invocation is at least $c' = (1 - c)^n$, which is a constant.

So, the probability for *MetHyl* not to terminate after t good invocations is at most $(1 - c')^t$. When $t \to +\infty$, this probability converges to 0. □

A.6 Proofs for Section 5.3

Theorem A.8 (Theorem 5.3)**.** *Given input program* (h, o) *where* h *is a relational hylomorphism and a set of instances* I*, let* p^* *be the program synthesized by MetHyl. Then* $\forall i \in I$*,* $(h, o) \sim_i p^*$*.*

Proof. Because the correctness of Step 4 can be proved in the same way as Step 2, this theorem is directly from Lemma 4.1, 4.3, 4.4, and the correctness of Step 4. □

Theorem A.9 (Theorem 5.4)**.** *Given input* $([\![\phi, \psi]\!]_{\mathsf{F}}, o)$ *and a grammar* G *specifying the program space for synthesis tasks, the program synthesized by MetHyl must be pseudo-polynomial time if the following assumptions are satisfied: (1)* ϕ*,* ψ *and programs in* G *runs in pseudo-polynomial time, (2) each value and the size of each recursive data structure generated by the input program are pseudo-polynomial, (3) all operators in* G *are linear, i.e., their outputs are bounded by a linear expression with respect to the input.*

Proof. The time complexity of the resulting program can be decomposed into four factors: (1) the number of recursive invocations on the generator, (2) the maximum number of partial solutions returned by each invocation, (3) the time complexity of each invocation of the generator, and (4) the time complexity of each invocation of the scorer. To prove this theorem, we only need to prove that all of these four factors are pseudo-polynomial.

First, we prove that for any program in G that returns a scalar value, its range is always pseudo-polynomial. For any such program p in G, let $f_p(n, w)$ be a polynomial representing that the time cost of p is at most $f_p(n, w)$ when n scalar values in range $[-w, w]$ are provided as the input.

By the third precondition, there exists a constant c such that for any operator $\oplus$ in G, for any input $\overline{x}$ and any output value $y \in \oplus \overline{x}$, $|y|$ is always at most $c \sum_{x \in \overline{x}} |x|$.

Suppose the size of program p is s_p, which is a constant while analyzing the complexity of p. Now, suppose n scalar values in range $[-w, w]$ are provided as the input to p. After executing the first operator, the sum of all available values is at most $f_p(n,w) \times cnw$, because there are at most $f_p(n,w)$ values due to the time limit and each value is at most cnw according to the third precondition. Then, after the second operator, this sum increases to $f_p(n,w) \times c(f_p(n,w) \times cnw) = c^2 f_p(n,w)^2 \times nw$. In this way, we know that after executing all s_p operators, the sum of all available values is at most $c^{s_p} f_p(n,w)^{s_p} \times nw$. Because s_p is a constant, this upper bound is still pseudo-polynomial with respect to the input.

Second, we prove that the first two factors are pseudo-polynomial. The first factor is bounded by the size of $?f_m$'s range, which is bounded by the product of the sizes of the ranges of key functions in $?f_m$. The second factor is bounded by the number of partial solutions returned by $thin\ ?R$. By Theorem 3.6, this value is also bounded by the product of the sizes of the ranges of key functions in $?R$. Because the number of key functions in $?f_m$ and $?R$ are constants, we only need to prove that the size of each key function's range is pseudo-polynomial.

- For key functions in $?f_m$, by the second precondition, in the input program, both the size of a state and values in a state are pseudo-polynomial with respect to the global input. By our first result, we obtain that the size of the range of each key function in $?f_m$ is pseudo-polynomial.
- For key functions in $?R$, by the second precondition, in the input program, both the size of a partial solution and values in a partial solution are pseudo-polynomial. By our first result, the scale of the new partial solution, i.e., the output of $?f_p$, must also be pseudo-polynomial. By the first result again, we obtain that the size of the range of each key function in $?R$ is pseudo-polynomial.

Third, we prove that the third factor is pseudo-polynomial. According to Section 4, the generator in the resulting program must be in the following form:

$$rg((thin\ ?R) \circ cup \circ \mathsf{P}\phi', \psi')$$

Therefore, the time complexity of each invocation can be further decomposed into four factors: (3.1) the time cost of $thin\ ?R$, (3.2) the time cost of ϕ', (3.3) the time cost of ψ', and (3.4) the number of invocations of ϕ'.

- According to Theorem 3.6, Factor 3.1 is bounded by the sizes of the ranges of the key functions in $?R$, which has been proven to be pseudo-polynomial.
- For Factor 3.1 (3.2), the time cost of ϕ' (ψ') is bounded by the time cost of ϕ (ψ) and all inserted program fragments $?q$ and $?c$ in Step 2 (Step 4). By the first precondition, their time costs are all pseudo-polynomial with respect to the new state (the new partial solution), which has also been proven to be pseudo-polynomial in both values and scale. Therefore, the time cost of ϕ' (ψ') is pseudo-polynomial.
- For Factor 3.3, by the first condition, the number of transitions (denoted as n_t) is pseudo-polynomial. The number of partial solutions returned by each recursive invocation (denoted as n_p) has been proven to be pseudo-polynomial, and the number of states (denoted by n_s) involved by a single transition is a constant. Therefore, the number of invocations of ϕ', which is bounded by $n_t \times n_p^{n_s}$, is also pseudo-polynomial.

Therefore, we prove that the third factor is also pseudo-polynomial with respect to the global input.

At last, the fourth operator is pseudo-polynomial because (1) the number of solutions and the scale of solutions are both pseudo-polynomial, and (2) the time complexity of the new objective function, which is a program in G, is pseudo-polynomial by the first precondition.

In summary, all four factors are pseudo-polynomial, and thus we prove the target theorem. □