Towards Efficient Inductive Synthesis of Expressions from Input/Output Examples

Jānis Bārzdiņš
Guntis Bārzdiņš
Kalvis Apsītis
Uğis Sarkans

Institute of Mathematics and Computer Science
University of Latvia
Rainis blvd. 29, Riga, LV-1459, Latvia
E-mail: jbarzdin@mii.lu.lv, guntis@mii.lu.lv

Abstract. Our goal through several years has been the development of efficient search algorithm for inductive inference of expressions using only input/output examples. The idea is to avoid exhaustive search by means of taking full advantage of semantic equality of many considered expressions. This might be the way that people avoid too big search when finding proof strategies for theorems, etc. As a formal model for the development of the method we use arithmetic expressions over the domain of natural numbers. A new approach for using weights associated with the functional symbols for restricting search space is considered. This allows adding constraints like the frequency of particular symbols in the expression. Additionally the current state of the art of computer experiments using this methodology is described. An example that is considered is the inductive inference of the formula for solving quadratic equations, the finding of which by pure exhaustive search would be unrealistic.

1 Introduction

Inductive synthesis of recursive functions from input/output examples is a very well studied problem in the recursive-theoretic framework [6] [7]. At the same time few works have been devoted to the problem in the practical perspective because it was considered impossible to synthesize non-trivial functions from input/output examples in the reasonable time (without exhaustive search).

One of the methods used to synthesize functions from input/output examples is the Occam razor principle stating that we have to search for the simplest hypothesis which complies with all available examples. This method could produce very reliable results, but it is difficult to be implemented without exhaustive search.

Nevertheless people are able to guess quite complicated functions from several input/output examples. Many such functions in the form of number sequences are collected in the Angluin's paper "Easily inferred sequences" [1]. The question is: what allows us to generalize such sequences so easily?

In [4] an idea to use algebraic axioms to synthesize functions from input/output examples was suggested. The main advantage of axioms is that they can be synthesized independently of each other and that the complexity of a separate axiom is much smaller than that of the whole program computing the function. If sufficiently many axioms are found, they can describe the function completely. The techniques known in the theory of term rewriting systems can be used to construct an executable program. The computer experiments have shown that algorithms for adding and multiplying binary numbers can be synthesized in this way. The most time-consuming part in such synthesis happens to be the synthesis of expressions in fixed signature which satisfy several input/output examples.

The problem of efficient synthesis of expressions from input/output examples presents interest also by itself. We might wish to be able to induce some formula, like the one for the volume of the frustum of a square pyramid:

$$V(h, a, b) = h\frac{a^2 + ab + b^2}{3}$$

using as input only the results of several measurements, e.g.:

$$\begin{aligned} V(6, 4, 2) &= 56 \\ V(3, 4, 3) &= 37 \\ V(9, 2, 1) &= 21 \\ V(6, 1, 3) &= 26 \end{aligned}$$

(This formula is particularly interesting because it was known in the ancient Egypt a long time before Euclid's deductive method in geometry was introduced [8]).

The studies on the problem of efficient inductive synthesis of expressions were initiated in [3] where a reasonably efficient method for such synthesis was proposed. The results of computer experiments illustrating this approach were described in [5]. These experiments have shown that the formulas like the one of the volume of the frustum of the square pyramid can be synthesized on the 33MHz Sparc workstation in about 10 minutes. The ultimate goal was to improve the method by an order of magnitude so that the formulas like the one for solving quadratic equations could be synthesized from input/output examples. In this case the method might become practically interesting.

In this paper a new, improved algorithm for inductive synthesis of expressions from input/output examples is presented. In many cases it might be more efficient than the one described in [3]. The first experimental results with this algorithm are given at the end of the paper.

2 Definitions and notations

Let the signature Σ be a finite set of functional symbols $\{f_1, \ldots, f_m\}$ where any symbol f_i has a fixed arity. Let $\mathcal{D}$ be a finite domain set. For the sake of simplicity we shall assume that $\mathcal{D}$ is a subset of natural numbers. We will say that Σ *is interpreted on* $\mathcal{D}$, if with all functional symbols $f \in \Sigma$ there is associated a partially defined function with domain and range in $\mathcal{D}$. By $K_{\Sigma,\mathcal{D}}$ we will denote a particular interpretation of Σ on $\mathcal{D}$. Since we are considering a finite domain $\mathcal{D}$, an interpretation $K_{\Sigma,\mathcal{D}}$ can be specified completely by a finite number of *equalities*. By equality we mean an expression

$$f(a_1, \ldots, a_n) = a_0$$

where $f \in \Sigma$ is a functional symbol of arity n and $a_0, a_1, \ldots, a_n \in \mathcal{D}$.

Example. Let $\Sigma_0 = \{z, s, +\}$ and $\mathcal{D}_0 = \{0, 1, 2, 3\}$. Then a particular interpretation $K_{\Sigma_0,\mathcal{D}_0}$ can be described by the following equalities:

$$\begin{aligned}
&\{z = 0, s(0) = 1, s(1) = 2, s(2) = 3,\\
&+(0,0) = 0, +(0,1) = 1, +(1,0) = 1, +(1,1) = 2, +(2,0) = 2,\\
&+(0,2) = 2, +(1,2) = 3, +(2,1) = 3, +(3,0) = 3, +(0,3) = 3\}.
\end{aligned}$$

For other parameter values functions are undefined in this interpretation.

Further by *Σ-algebra* $\mathcal{K}$ we shall understand the set of equalities as in the Example, which describes an interpretation $K_{\Sigma,\mathcal{D}}$ over fixed domain $\mathcal{D}$. We assume that $\mathcal{D}$ contains only those domain elements which appear on left or right side of some equality from $\mathcal{K}$. The *volume* of Σ-algebra $\mathcal{K}$ is defined to be the number of equalities in $\mathcal{K}$ and will be denoted $|\mathcal{K}|$. Σ_0-algebra $\mathcal{K}_0$ in the Example has volume $|\mathcal{K}_0| = 14$.

Let there be a fixed alphabet $\{x_1, x_2, \ldots\}$ of *term variables*. *Open terms* are expressions made of term variables and functional symbols from the signature Σ.

By *weight function* we shall understand a mapping from the set of all open terms to the set of all nonnegative integers: $w : T(\Sigma) \to N$ with certain properties. Namely, weight function w is defined by means of *auxiliary weight functions* $\{\tilde{f}_1, \ldots, \tilde{f}_m\}$ where $\tilde{f}_j : N^n \to N$ is of the same arity as f_j:

- Any variable x_i is an open term of weight $w(x_i) = 0$.
- Any 0-arity functional symbol is an open term of constant weight

$$w(f()) = \tilde{f}$$

- If f is a functional symbol of arity n $(n > 0)$ and $t_1, \ldots, t_n$ are open terms, then the expression $f(t_1, \ldots, t_n)$ is an open term of weight

$$w(f(t_1, \ldots, t_n)) = \tilde{f}(w(t_1), \ldots, w(t_n))$$

Auxiliary weight functions $\tilde{f}_j$, $j \in \{1, \ldots, m\}$, must satisfy monotonicity axioms:

A1. $\tilde{f}_j(p_1, \ldots, p_n) > \max(p_1, \ldots, p_n)$.

A2. If $p'_i \geq p_i$ for each $i \in \{1, \ldots, n\}$ then $\tilde{f}_j(p'_1, \ldots, p'_n) \geq \tilde{f}_j(p_1, \ldots, p_n)$.

Besides that functions $\tilde{f}_j$, $j \in \{1, \ldots, m\}$, must be computable from its arguments in constant time.

Some examples of weight function w (over signature $\Sigma_0 = \{z, s, +\}$):

- $\tilde{z} = 1, \tilde{s}(p) = p + 1, \tilde{+}(p_1, p_2) = \max(p_1, p_2) + 1$ (in this case $w(t)$ describes the number of levels in the term),
- $\tilde{z} = 1, \tilde{s}(p) = p + 1, \tilde{+}(p_1, p_2) = p_1 + p_2 + 1$ (in this case $w(t)$ is number of functional symbols in the term).

Let weight function w is arbitrary fixed. Further by *weight* of open term t we will understand $w(t)$.

The *size* of the open term t is defined to be the number of instances of the functional and variable symbols in t and will be denoted $|t|$.

The term obtained from the open term t by replacing its variables by elements of domain set $\mathcal{D}$ will be called a *closed term*, and its weight and size are defined to be the same as the weight and size of the open term t.

We will say that a closed term t can be computed in Σ-algebra $\mathcal{K}$ if its value can be derived by means of elementary equations of $\mathcal{K}$.

Let there be given a tuple $\langle a_1, \dots, a_k \rangle \in \mathcal{D}^k$, $a_i \neq a_j$ if $i \neq j$, and $b \in \mathcal{D}$.[1] Such pair

$$(\langle a_1, \dots, a_k \rangle, b)$$

we will call *input/output example.* We say that an open term t *satisfies* the I/O-example $(\langle a_1, \dots, a_k \rangle, b)$ in the Σ-algebra $\mathcal{K}$ if:

- t contains no other variables than $x_1, \dots, x_k$,
- the value of a closed term t' obtained from t by replacing variables $x_1, \dots, x_k$ by $a_1, \dots, a_k$ respectively, can be computed in $\mathcal{K}$,
- the value of t' is equal to b.

Let l be a natural number ($l > 0$). We will denote by $A_{\mathcal{K},l}^{(\langle a_1,\dots,a_k \rangle, b)}$ a set of all open terms such that:

- they satisfy the input/output example $(\langle a_1, \dots, a_k \rangle, b)$ in $\mathcal{K}$,
- they have weight no more than l.

If we consider the Σ_0-algebra $\mathcal{K}_0$ and the weight function $w(t)$ equal to the number of functional symbols in t, then the set $A_{\mathcal{K}_0,2}^{(\langle 1,2 \rangle, 3)}$ is

$$\{+(x_1, x_2),\ +(x_2, x_1),\ +(x_1, s(x_1)),\ +(s(x_1), x_1),\ +(x_1, +(x_1, x_1)),\\ +(+(x_1, x_1), x_1),\ s(x_2),\ s(s(x_1)),\ s(+(x_1, x_1))\}.$$

3 Synthesis from one example

We will say that an algorithm having received the input U enumerates the set of objects $\{w_1, w_2, \dots, w_s\}$ in *setup time* T and ith *step time* T_i, if this algorithm outputs ("prints" on the output tape) the first object w_1 in $T + T_1$ time, and the ith object $w_i (i = 2, 3, \dots)$ in T_i time from the moment when the previous object w_{i-1} was output. In this paper by *algorithm* we mean the RAM-machine.

[1] The restriction that $a_i \neq a_j$ if $i \neq j$ is not essential — it is added only to simplify the algorithm.

Theorem 1. *Let signature Σ and weight function w be fixed. There exists an algorithm which, given any Σ-algebra $\mathcal{K}$, any input/output example $(\langle a_1, \dots a_k\rangle, b)$ and any natural number $l > 0$, enumerates without repeating the set of terms $A_{\mathcal{K},l}^{(\langle a_1,\dots a_k\rangle,b)}$ in setup time $O(|\mathcal{K}|\log l)$ and ith-step time $O(|t_i|\log l)$, where t_i is the output term in the ith step $(i = 1, 2, \dots, |A_{\mathcal{K},l}^{(\langle a_1,\dots a_k\rangle,b)}|)$.*

Proof. For Σ-algebra $\mathcal{K}$ we define a corresponding Σ-algebra graph $G_{\mathcal{K}}$. Σ-algebra graph $G_{\mathcal{K}}$ contains nodes of two types: domain nodes denoted by $D_{\mathcal{K}}$ and functional nodes denoted by $F_{\mathcal{K}}$. To distinguish nodes of these two types, in the figures we will show domain nodes as dots and functional nodes as small circles. Domain nodes will correspond to the elements of domain set $\mathcal{D}$ of Σ-algebra $\mathcal{K}$. Functional nodes will correspond to elementary equations of Σ-algebra $\mathcal{K}$ and will be marked by the functional symbol on the left-hand side primitive term in this equation. For any equation

$$f(a_1, \dots, a_n) = b$$

of Σ-algebra $\mathcal{K}$ the following arcs are added in graph $G_{\mathcal{K}}$. From a functional node v corresponding to this equation an arc is drawn to the domain node b; the node b is called the *upper node* for the functional node v. From domain nodes $a_1, \dots, a_n$ arcs (marked by numbers $1, 2, \dots, n$ respectively) are drawn to the functional node v; domain nodes $a_1, \dots, a_n$ are called *lower nodes* for the functional node v.

The Σ_0-algebra graph $G_{\mathcal{K}_0}$ which corresponds to the Σ_0-algebra $\mathcal{K}_0$ given above is shown in Fig.1.

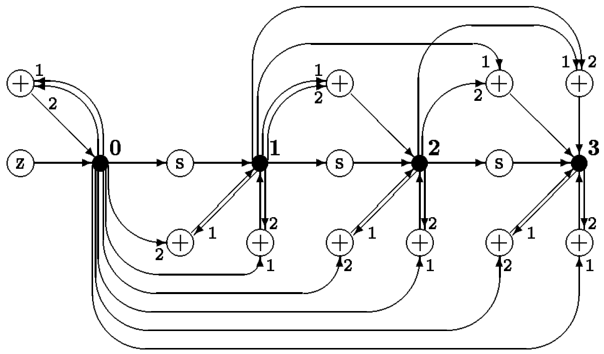

Fig. 1. Σ_0-algebra graph $G_{\mathcal{K}_0}$.

Let there be some tuple $\langle a_1, \dots, a_k\rangle \in \mathcal{D}^k$ such that $a_i \neq a_j$ if $i \neq j$. Let there be some natural number l. In this case we define weights for nodes of the graph $G_{\mathcal{K}}$ according to the following conditions (weight will not be defined for all nodes):

- Domain nodes $a_1, \ldots, a_k$ have weight 0.
- Any functional node which corresponds to 0-arity functional symbol f has weight $\tilde{f}$.
- If a functional node v corresponds to some n-arity functional symbol f and $d_1, \ldots, d_n$ are lower nodes of the node v and for all of them weights $h_1, \ldots, h_n$ are defined, then the weight of the node v is $\tilde{f}(h_1, \ldots, h_n)$; otherwise the weight for the node v is not defined.
- Let $v_1, \ldots, v_n$ be functional nodes for which the upper domain node is d and weights $h_1, \ldots, h_n$ are defined. In this case the weight of the domain node d is $\min(h_1, \ldots, h_n)$; otherwise the weight of the domain node d is not defined (i.e. if no such functional nodes v_i do exist).
- All weights do not exceed l.

It is easy to see that a weight related this way to some domain node d is exactly the smallest weight among weights of closed terms which in Σ-algebra $\mathcal{K}$ satisfy the pair $(\langle a_1, \ldots, a_k\rangle, d)$, if it does not exceed l. (According to axiom A2, if $t = f(t_1, \ldots, t_k)$ and minimal weights of terms $t_1, \ldots, t_k$ are, respectively, $\tilde{t_1}, \ldots, \tilde{t_k}$, minimal weight of t is

$$\tilde{t} = \tilde{f}(\tilde{t_1}, \ldots, \tilde{t_k}).)$$

Additional arcs, called *dotted arcs* (they are denoted by dotted lines), are added to the graph $G_{\mathcal{K}}$ in the following way. Let d be some domain node for which weight is defined. Let $v_1, \ldots, v_n$ be functional nodes whose upper node is d and for which weights are defined and let these nodes $v_1, \ldots, v_n$ be already ordered according to their weights ($\mathrm{weight}(v_i) \leq \mathrm{weight}(v_{i+1})$). The dotted arc is drawn to connect the node d to v_1, the node v_1 to v_2, ..., the node v_{n-1} to v_n.

The graph $G_{\mathcal{K}}$ which is completed by weights and dotted arcs according to the fixed tuple $\langle a_1, \ldots, a_k\rangle$ and fixed l will be called *an annotated Σ-algebra graph* and will be denoted $G_{\mathcal{K},l}^{\langle a_1,\ldots,a_k\rangle}$.

The graph $G_{\mathcal{K}_0,3}^{\langle 1,2\rangle}$ which corresponds to the Σ_0-algebra graph $G_{\mathcal{K}_0}$ is shown in Fig.2.

3.1 Construction of annotated sample graph

Lemma 2. *The annotated Σ-algebra graph $G_{\mathcal{K},l}^{\langle a_1,\ldots,a_k\rangle}$ can be built from Σ-algebra $\mathcal{K}$ in time $O(|\mathcal{K}|\log l)$.*

Proof. The algorithm consists of initial step, iterative step and final step.

Initial step. For every domain node $a_i, i = 1, \ldots, k$, we introduce an additional 0-arity functional node with the upper node a_i and set its weight equal to zero. For every "proper" 0-arity functional node f_i we set the weight to $\tilde{f_i}$, if and only if $\tilde{f_i} \leq l$.

Iterative step. First let us examine the graph after the ith iterative step. (We will refer to the initial step as the 0-th step.) There are four kinds of functional nodes.

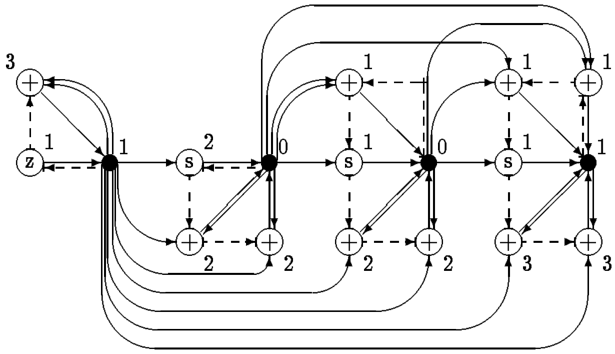

Fig. 2. Annotated Σ_0-algebra graph $G^{\langle 1,2\rangle}_{\mathcal{K}_0,3}$.

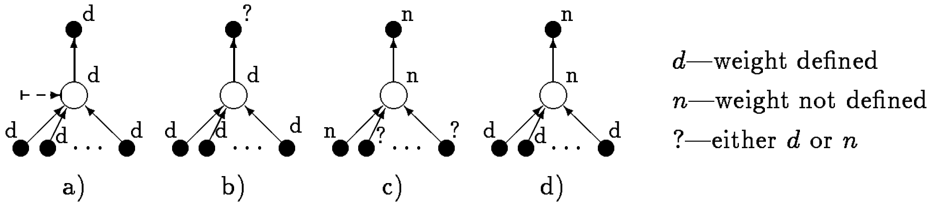

Fig. 3. Types of functional nodes after iterative step.

- Node with a dotted arc entering it. Weights of its lower nodes, upper node and itself are defined (Fig. 3a). We denote the set of all such nodes by A_i.
- Node without a dotted arc entering it, but with defined weight. Weights of the lower nodes are defined (Fig. 3b). The set of all such nodes is denoted by B_i.
- Node with undefined weight. The weight of the upper node is also undefined. At least one lower node has undefined weight (Fig. 3c). The set of all such nodes will be called C_i.
- Node with undefined weight, but all lower nodes has defined weight (Fig. 3d). The corresponding set is denoted by D_i.

After the initial step only B_0 and C_0 are not empty.

And now the iterative step itself.

While B_i is not empty we take from it the functional node with minimal weight (any, if there are several ones) f_i. Two cases should be distinguished.

1. The weight of f_i upper node d_i is defined. Then we add f_i to the end of the dotted path associated with d_i.

$$B_{i+1} = B_i \setminus \{f_i\}, A_{i+1} = A_i \cup \{f_i\}, C_{i+1} = C_i, D_{i+1} = D_i$$

2. The weight of f_i upper node d_i is not defined. We set it equal to the weight of f_i, draw a dotted arc from d_i to f_i.

$$A_{i+1} = A_i \cup \{f_i\}$$

Afterwards we examine all functional nodes with d_i as a lower node (they belong to C_i). Suppose there are k_i functional nodes $g_{i,0}, \ldots, g_{i,k_i}$ that have only lower nodes with defined weights. (For any node $g_{i,j}, j = 0, \ldots, k_i$, d_i was the last node with undefined weight.) Now we can compute $\tilde{g}_{i,0}, \ldots, \tilde{g}_{i,k_i}$. Let us suppose that there is l_i such that for any $j = 0, \ldots, l_i$, $\tilde{g}_{i,j} \leq l$, and for any $j = l_{i+1}, \ldots, k_i$, $\tilde{g}_{i,j} > l$. We set weights for nodes $g_{i,0}, \ldots, g_{i,l_i}$.

$$B_{i+1} = (B_i \setminus f_i) \cup \{g_{i,j} | j = 0, \ldots, l_i\},$$

$$C_{i+1} = C_i \setminus \{g_{i,j} | j = 0, \ldots, k_i\},$$

$$D_{i+1} = D_i \cup \{g_{i,j} | j = l_i + 1, \ldots, k_i\}.$$

It is important to notice that, if every node in A_i has weight not greater than any node in B_i has, the same property holds also for A_{i+1} and B_{i+1} (because f_i is the node with the minimal weight and for every $j = 0, \ldots, k_i$, $\tilde{g}_{i,j} > \tilde{f}_i$ — axiom A1). Therefore every dotted path is correct, i. e. nodes along a path are ordered according to their weights.

Final step. We remove additional 0-arity functional nodes that were added during initial step. (It is necessary to delete them correctly from the dotted paths.)

We must be able to find a functional node in B_i with minimal weight. We will organize set B_i so that it would be possible to insert, delete and find a functional node with minimal weight in time $O(\log l)$. For this purpose a priority queue [2] will be used.

Let us analyze time complexity of our algorithm.

Initial step. The number of domain nodes and 0-arity functional nodes is $O(|\mathcal{K}|)$, adding to B_0 takes time $O(\log l)$ for every node. Therefore initial step takes time $O(|\mathcal{K}| \log l)$.

Iterative step. The number of iterative steps is $O(|\mathcal{K}|)$. Although an individual step may require time greater than $O(\log l)$ (if there are many functional nodes with d_i as a lower node), total time required by iterative steps is $(O(|\mathcal{K}| \log l)$. (Every node can be moved from C_i to B_{i+1} only once and from B_i to A_{i+1} also only once.)

Final step. It takes time $O(|\mathcal{K}|)$. □

3.2 Term output

Let there be a graph $G_{\mathcal{K},l}^{(a_1,\ldots,a_k)}$ placed in memory so that its nodes can be accessed by their addresses.

A closed term will be called an *α-term* if:

– it contains no other domain symbols than $a_1, \ldots, a_k$,
– it belongs to the Σ-algebra $\mathcal{K}$.

An α-term will be called *annotated* if, for each of its symbols, the address of some node in the graph $G_{\mathcal{K},l}^{\langle a_1,\ldots,a_k\rangle}$ is attached according to the following conditions:

– to domain symbols there are attached the addresses of the corresponding domain nodes,
– if $f(g_1, \ldots, g_n)$ is a subterm of a term t and $v_1, \ldots, v_n$ are domain nodes which represent values of closed terms $g_1, \ldots, g_n$, then to the symbol f there is attached the address of the functional node marked by f and having lower nodes $v_1, \ldots, v_n$.

It is easy to see that any α-term can be annotated.

We shall use two representations of α-terms— ordered tree form and linear form (in prefix notation). Subterms of the same level which are included in brackets of the same functional symbol (their parent in the tree) will be called siblings. Siblings are ordered according to the number of argument they represent. Therefore we can speak of right siblings for given subterm. (Such trees and linear layouts for current α-terms will be stored apart from the Σ-algebra graph $G_{\mathcal{K},l}^{\langle a_1,\ldots,a_k\rangle}$.)

We define the *minimal α-term* of a domain node d in the following way. If $d \in \{a_1, \ldots, a_k\}$, then the symbol d itself is the minimal α-term of the node d. Otherwise we consider the functional node v which is connected to the node d by a dotted arc (if there is no dotted arc from the node d, then the minimal term is not defined). If the node v corresponds to 0-arity functional symbol f, then this symbol itself is the minimal term. If the node v corresponds to n-arity ($n > 0$) functional symbol f, then the minimal term is the term $f(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are minimal α-terms of the lower nodes for the node v.

It is easy to see that the so defined minimal α-term of the node d is a closed term which satisfies the pair $(\langle a_1, \ldots, a_k\rangle, d)$ in Σ-algebra $\mathcal{K}$ and has the minimal weight which is the weight of the node d in the Σ-algebra graph.

Now we also define the minimal α-term for functional nodes. Let v be some functional node for which weight is defined (otherwise the minimal term for v is not defined). Let functional node v correspond to the functional symbol f of arity n. If $n = 0$, then the symbol f itself is the minimal term of node v. Otherwise there are lower nodes $d_1, \ldots, d_n$ for the node v and minimal terms $t_1, \ldots, t_n$ for those domain nodes. Then the term $f(t_1, \ldots, t_n)$ is the minimal term of the functional node v. It is easy to see that the weight of the minimal term of the functional node v is equal to the weight of the node v in the Σ-algebra graph.

Lemma 3. *Inequality*

$$\tilde{f}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_n) \leq l$$

can be solved in time $O(\log l)$ and solution, if it exists, is in form

$$x \leq l'$$

($p_1, \ldots, p_n$,l,l',x are natural numbers, $l' < l$).

Indeed, A2 implies that together with l' all $x < l'$ suit inequality as well. A1 implies that l' (if it exists at all) should be smaller than l. The exact value of l' can be found in $O(\log l)$ steps of dichotomy.[2]

For the representation of a given α-term we attach *actual* and *maximal weights* to every ordered tree node (or, equivalently, to every instance of functional or variable symbol) in the following way:

- for the root symbol maximal weight l is given,
- if a parent node f has its maximal weight l', then its i-th child node has maximal weight l'', where $x \leq l''$ is the solution of inequality

  $$\tilde{f}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_n) \leq l'$$

  ($p_1, \ldots, p_n$ are actual weights of sibling nodes). In the case solution does not exist we assume $l'' = -1$.

Maximal weight shows how far we can raise the weight of a given subterm without making other changes.

Actual weights are computed upwards in $O(|t|)$ time, and after that — maximal weights are computed downwards in time $O(|t| \log l)$.

We will say that there exists an *l'-alternative* for annotated α-term t in the graph $G_{\mathcal{K},l}^{(a_1,\ldots,a_k)}$ if from the node q_1 there is a dotted arc to the node q_2 of weight l'. In this case the minimal α-term of the node q_2 is called the *l'-alternative* of term t.

The following *ordering of subterms* for a term t is defined: we say that a subterm t_1 is *left* from the subterm t_2 if the root symbol of t_1 is left from the root symbol of t_2 in the usual linear layout of the term t.

We mark the current α-term in the following way. A special symbol, say "$*$", is associated with the rightmost subterm t_1 of the term t (according to the ordering of subterms defined above) for which the l'-alternative exists, where l' is not greater than the maximal weight assigned to this subterm in the tree representation of t.

It is easy to see that for marking of the annotated α-term t, or detecting it as unmarkable, $O(|t|)$ steps are necessary.

For marked α-term t we define *succeeding α-term t'* in the following way. Let t_1 be the subterm marked by "$*$" and t_2 be the alternative of the term t_1. Succeeding term is obtained from the term t by replacing its subterm t_1 by the subterm t_2 and all its right siblings by the minimal terms of the same value.

The next lemma follows easy from the facts already proved.

Lemma 4. *The number of the steps necessary to construct, from a marked α-term t, its succeeding α-term t' and to mark it is $O(|t'| \log l)$.*

[2] Some important weight functions allow solving the inequality in constant time

The algorithm mentioned in Theorem 1 works as follows. Having received as input the Σ-algebra $\mathcal{K}$, the pair $(\langle a_1, \ldots, a_k \rangle, b)$ and $l > 0$, it constructs first the graph $G_{\mathcal{K},l}^{\langle a_1,\ldots,a_k \rangle}$. It takes $O(|\mathcal{K}| \log l)$ steps. After that the algorithm outputs the minimal term t_1 of the node b (in which the different domain values $a_1, \ldots, a_k$ are replaced by variables $x_1, \ldots, x_k$ respectively); this takes $O(|t_1|)$ steps. Then for the term t_2, succeeding t_1, the corresponding open term is output (according to the last lemma it takes $O(|t_2|)$ steps), etc. We proceed while marking is possible. It is easy to see that in this way the algorithm enumerates the set $A_{\mathcal{K},l}^{(\langle a_1,\ldots,a_k \rangle, b)}$ without repeating in the time mentioned in Theorem 1.□

4 Synthesis from several examples

In the previous section the case where an open term satisfies one input/output example

$$(\langle a_1, \ldots, a_k \rangle, b)$$

was considered. Now we will consider the case where an open term has to satisfy several input/output examples simultaneously. More precisely, let there be a set of input/output examples belonging to Σ-algebra $\mathcal{K}$:

$$Q = \{(\langle a_1^1, \ldots, a_k^1 \rangle, b^1), (\langle a_1^2, \ldots, a_k^2 \rangle, b^2), \ldots, (\langle a_1^s, \ldots, a_k^s \rangle, b^s)\}.$$

We will say that an open term t *satisfies* a set Q in Σ-algebra $\mathcal{K}$, if t satisfies all elements of Q in $\mathcal{K}$.

Let us denote by $A_{\mathcal{K},l}^{Q}$ the set of all open terms which in Σ-algebra $\mathcal{K}$ satisfy a set Q and have weight no more than l $(l > 0)$. In other words,

$$A_{\mathcal{K},l}^{Q} = A_{\mathcal{K},l}^{(\langle a_1^1,\ldots,a_k^1 \rangle, b^1)} \cap \ldots \cap A_{\mathcal{K},l}^{(\langle a_1^s,\ldots,a_k^s \rangle, b^s)}.$$

In real situations it is typical that separate sets

$$A_{\mathcal{K},l}^{(\langle a_1^1,\ldots,a_k^1 \rangle, b^1)}, A_{\mathcal{K},l}^{(\langle a_1^2,\ldots,a_k^2 \rangle, b^2)}, \ldots$$

are of relatively large size but their intersection is of relatively small size. Therefore an important question arises: can elements of $A_{\mathcal{K},l}^{Q}$ be found directly (i.e. enumerated sufficiently fast) without constructing all s sets?

The next theorem gives a positive answer to this question in some sense.

Theorem 5. *Let signature Σ and weight function w be fixed. There exists an algorithm which, given any Σ-algebra $\mathcal{K}$, any set Q of input/output examples and any natural number $l > 0$, enumerates without repeating the set of terms $A_{\mathcal{K},l}^{Q}$ in setup time $O(|\mathcal{K}|^{|Q|})$ and ith-step time $O(|t_i|)$ where t_i is the output term in the step i $(i = 1, 2, \ldots, |A_{\mathcal{K},l}^{Q}|)$.*

Proof. For the sake of simplicity we will consider the case when the set Q contains only two examples: $(\langle a_1^1, \ldots, a_k^1 \rangle, b^1)$ and $(\langle a_1^2, \ldots, a_k^2 \rangle, b^2)$.

For Σ-algebra $\mathcal{K}$ we define a *square Σ-algebra* $\mathcal{K}^2$ which describes an interpretation $K_{\Sigma,\mathcal{D}^2}$:

$$(f_i((a_1, a_1'), \ldots, (a_n, a_n')) = (b, b')) \in \mathcal{K}^2$$

if and only if

$$(f_i(a_1, \ldots, a_n) = b) \in \mathcal{K} \text{ and } (f_i(a_1', \ldots, a_n') = b') \in \mathcal{K}.$$

It is easy to deduce the next important equality:

$$A_{\mathcal{K},l}^{\{(\langle a_1^1,\ldots,a_k^1\rangle,b^1),(\langle a_1^2,\ldots,a_k^2\rangle,b^2)\}} = A_{\mathcal{K},l}^{(\langle a_1^1,\ldots,a_k^1\rangle,b^1)} \cap A_{\mathcal{K},l}^{(\langle a_1^2,\ldots,a_k^2\rangle,b^2)} =$$

$$= A_{\mathcal{K}^2,l}^{(\langle(a_1^1,a_1^2),\ldots,(a_k^1,a_k^2)\rangle,(b^1,b^2))}$$

To enumerate the set $A_{\mathcal{K},l}^{\{(\langle a_1^1,\ldots,a_k^1\rangle,b^1),(\langle a_1^2,\ldots,a_k^2\rangle,b^2)\}}$ the algorithm described in Theorem 1 can be applied with respect to Σ-algebra $\mathcal{K}^2$. Because $|\mathcal{K}^2| \leq |\mathcal{K}|^2$ the correctness of Theorem 5, when $|Q| = 2$, follows immediately. □

5 Computer experiments

The advantage of the given algorithm (if compared with the algorithm described in [3]) is that it allows to enumerate effectively not only terms having limited depth, but it also permits to enumerate terms according to more sophisticated criteria, like the number of functional symbols etc. In the case some function (like square root) is not likely to appear in the expression more than once, then the new method allows to restrict that by assigning higher weight to the particular function.

First some implementation details. Setup time $O(|\mathcal{K}|^{|Q|})$ and corresponding volumes of graphs still are too large for practical implementations. Another approach was used — not only the weight of terms was restricted with some w, but also the term level was said not to exceed some l (it is easy to see that the method described above works also for such a pair of restrictions). Besides, the annotated sample graph was constructed dynamically, it contained only reachable domain nodes. Reachability of some node here means that

- there is a term of depth not greater than l such that, substituting its variables with values from input example, the value of the term corresponds to the node;
- the weight of this term does not exceed w;
- there is a term of depth not greater than l such that, substituting some of its variables with the value corresponding to the node and other variables with values corresponding to reachable nodes, the value of the term equals to the value of output example;

– the weight of this term also does not exceed w. (Note that there is only one input example consisting of several tuples and only one output example (tuple) — see Section 4).

Using described approach a formula for the volume of the frustum of the square pyramid was synthesized. The experiment showed that the new algorithm finds the formula several times faster than the algorithm described in [5].

The next formula we tried to synthesize was the formula for finding the greatest root of quadratic equation of the form $x^2+bx+c=0$. Unfortunately we were not able to synthesize the formula from input/output examples using 33MHz Sparc workstation in the reasonable time, assuming $\Sigma = \{+,-,*,/,\sqrt{\ },1,2,3,4\}$ and using weight function which counts instances of functional symbols . Therefore the algorithm of Theorem 2 was further modified taking into account the following observation: the outermost function of the formula is one of the 9 functions appearing in Σ and having fixed one of them we can reduce the depth of unknown term by 1 level (we have to consider all 9 possible cases of the outermost function). In fact it is reasonable to fix functions appearing in the two outermost levels what yields us 377 cases, but reduces the depth of the unknown term by 2 levels. In this way the formula for greatest root of the quadratic equation

$$x = \frac{\sqrt{b^2-4c}-b}{2}$$

was synthesized on the 33MHz Sparc workstation in 20 minutes from the following examples:

$$\begin{aligned} F(3,-4) &= 1 \\ F(-2,-3) &= 3 \\ F(5,4) &= -1 \\ F(-4,3) &= 3 \\ F(-3,-4) &= 4 \\ F(-3,2) &= 2 \\ F(3,2) &= -1 \end{aligned}$$

Limit on the number of functional symbols — 8, limit on term depth — 6, domain — [–16,25]. Analogical attemt without weight restriction failed.

Then we changed domain for [–24,36], and examples were as follows:

$$\begin{aligned} F(-1,-6) &= 3 \\ F(4,-5) &= 1 \\ F(5,6) &= -2 \\ F(-4,-5) &= 5 \\ F(6,5) &= -1 \\ F(-5,4) &= 4 \end{aligned}$$

We expected to obtain greater annotated sample graphs, as the domain was greater, but surprisingly the correct formula was synthesised in 10 minutes with much smaller graph volumes. Appearently these examples were "better" if compared to previous ones. Therefore it seems to be hard to make any theoretical estimations of the behavior of annotated sample graphs on various examples.

Further experiments are intended in order to more precisely understand the power of the described method.

References

1. D.Angluin: Easily inferred sequences. Memorandum No. ERL-M499, University of California, 1974
2. P.van Emde Boas, R.Kaas and E.Zijlstra: Design and Implementation of an Efficient Priority Queue. Mathematical Systems Theory 10 (1977), pp. 99—127
3. J.M.Barzdins and G.J.Barzdins: Rapid construction of algebraic axioms from samples. Theoretical Computer Science 90 (1991), pp. 199—208
4. G.Barzdins: Inductive synthesis of term rewriting systems. Lecture Notes in Computer Science, vol.502 (1991), pp. 253—285
5. J.Barzdins and G.Barzdins: Towards efficient inductive synthesis: rapid construction of local regularities. Lecture Notes in Computer Science (to appear).
6. R.Freivalds: Inductive inference of recursive functions. Qualitative theory. Lecture Notes in Computer Science, vol.502 (1991), pp. 77—110
7. R.Freivalds, J.Barzdins, K.Podnieks: Inductive inference of recursive functions: Complexity bounds. Lecture Notes in Computer Science, vol.502 (1991), pp. 565—613
8. E.Howard: An Introduction to the History of Mathematics. New York: Holt, Reinhard and Winston 1961

This article was processed using the LaTeX macro package with LLNCS style