Automatic Synthesis of Merging and Inserting Algorithms on Binary Trees using Multisets in *Theorema*

Isabela Drămnesc[1] and Tudor Jebelean[2]

[1] West University, Timişoara, Romania
isabela.dramnesc@e-uvt.ro
[2] Johannes Kepler University, Linz, Austria
Tudor.Jebelean@jku.at

Abstract. We demonstrate the automatic proof–based synthesis of merging and inserting algorithms for [sorted] binary trees, using the notion of multisets, in the *Theorema* system. Each algorithm is extracted from the proof of the conjecture based on the specification of the desired function, in the form of a list of [conditional] equalities, which can be directly executed. The proofs are performed in natural style, using general techniques, but most importantly efficient inference rules and strategies specific for the domains involved. In particular we present specific techniques for the construction of arbitrarily nested recursive algorithms by general Noetherian induction, as well as a systematic method for the generation of the conjectures and consequently of the algorithms for the auxiliary functions needed in the main function.

Keywords: algorithm synthesis · binary trees · multisets · Theorema

1 Introduction

Automated synthesis of algorithms based on logical principles is an interesting alternative to algorithm verification, because it focuses on the study of the properties of the involved domains, from which correct algorithms are obtained automatically, instead of creating them by human ingenuity. The case studies presented in this paper are part of our research on systematic theory construction (*theory exploration* [2]) and automated synthesis in the domain of finite *binary trees* for which we also use finite *multisets*. In two related papers [9, 10] we already investigated algorithms for deletion from lists and binary trees, as well as sorting algorithms for lists. Multisets allow to express in a natural way the fact that two trees have the same elements, but more importantly (as are revealed by our experiments) it leads to powerful proof techniques. For space reasons, in this presentation we focus on *one argument induction*[3] and also on

[3] For binary functions one may use simultaneous induction on both arguments.

compositional construction[4] and do not approach yet algorithms which use both lists and trees. We approach automated synthesis as described in our previous work – see e.g. [8,15]. First one proves automatically a *synthesis conjecture* which is based on the *specification* (input and output conditions) of the desired function, then the algorithm is extracted automatically from the proof. We use the *Theorema* system [6], in which the inference rules and the logical formulae are presented in *natural style* – similar to the one used by humans. Since *Theorema* also allows the execution of algorithms, we can test them immediately in the system. The theoretical basis and the correctness of the proof based synthesis scheme is well–known, see [7, 18].

Each algorithm is produced as a list of clauses, each clause being a (possibly conditional) universally quantified equality which is to be applied as a rewrite rule from left to right. The LHS[5] of each equality consists of the function symbol (of the desired algorithm) applied to a term which identifies a certain class of possible inputs (this is sometimes called *pattern matching* programming). The clauses are such that all possible inputs are considered (*covering*), and no two clauses may apply to the same input (*mutual exclusion*) – these properties are automatically insured by the synthesis method.

Related work and originality. [18] introduces deductive techniques for algorithm synthesis, in particular for constructing recursive algorithms. These techniques are applied in [23] to manually derive several sorting algorithms in the theories of integers and strings. They present also a rule for generating auxiliary algorithms, see also [20]. Later implementations using some of these principles are in [17, 22]. We presented a more detailed survey of synthesis methods in [8]. In the current paper we follow some of the principles from [18,23], but we develop different proof–based techniques for algorithm synthesis.

The theory of *multisets* is well studied in the literature, including computational formalizations (see e. g. [19], where finite multisets are called *bags*). A presentation of the theory of multisets and a good survey of the literature related to multisets and their usage is [1] and some interesting practical developments are in [21]. In previous work on synthesis, multisets are not explicitly used in the process of proof–based algorithm synthesis. They are just mentioned in the problem specification (e.g., in expressing the permutation of two objects), but their definition and properties are not involved in the process of proof–based algorithm synthesis. In this paper we explicitly use multisets, their definition and properties in the entire process of algorithm synthesis.

In our previous work we study proof–based algorithm synthesis in the theories of lists, sets and binary trees [12] separately ([14], [8], [15]), but without using multisets.

A systematic formalization of the theory of lists using multisets for the proofs of correctness of various sorting algorithms is mechanized in Isabelle/HOL[6],

[4] The construction of the object desired for the synthesis uses only the objects which are already present in the proof, and does not try to decompose some of them.

[5] We use LHS for left hand side and RHS for right hand side.

[6] https://isabelle.in.tum.de/library/HOL/HOL-Library/Sorting_Algorithms.html

but this does not address the problem of algorithm synthesis. An interesting formalization in a previous version of *Theorema* [5], which includes the theory exploration and the synthesis of a sorting algorithm is presented in [4], which also constituted the inspiration of our previous research on proof–based synthesis. However, in that pioneering work, the starting point of the synthesis (besides the specification of the desired function) is a specific *algorithm scheme*, while in our approach we use induction principles and dynamic induction.

In contrast to other investigations and to our previous research, the current study *uses multisets* in the synthesis problem and in the entire process of algorithm synthesis, combined properties which are necessary in the process of algorithm synthesis, the automatically generated proofs are performed in the new version of the *Theorema* system [6, 24], and the investigation is performed in parallel on the two domains. We already investigated the proof–based synthesis of auxiliary algorithms on binary trees: *Merge* [11], and *Insert* [13], see also [15], but we did not use multisets and we applied different proof techniques. In our current approach using multisets we investigate in companion papers the synthesis of *Delete* on lists and trees [9] and the synthesis of sorting algorithms on lists [10].

Moreover, this paper describes more precisely the practical techniques for *cascading* and for *general Noetherian induction*, and illustrates them in more detail on several examples. For the purposes above, **three novel inference rules** are introduced, and **seven inference rules** and **seven strategies** are extended for these case studies on binary trees using multisets.

2 Proof–Based Synthesis

2.1 Context

Notations. We use square brackets for function and for predicate application, for instance: $f[x]$ instead of $f(x)$ and $P[a]$ instead of $P(a)$. Quantified variables are placed under the quantifier, as in $\underset{X}{\forall}$ and $\underset{X}{\exists}$.

The objects occurring in the formulae are: *elements* — objects from a totally ordered domain (denoted a, b, c) which are members of composite objects; *multisets* denoted A, B, C; and *binary trees* denoted L, R, S, T, X, Y, Z. (Multisets and binary trees are also addressed as *composite objects*).

Knowledge base. For space reasons, we list explicitly only the formulae which are used in the proofs presented in this paper, the theory exploration includes more statements.

Elements of various composite structures are any objects whose domain is totally ordered (notation $\leq$ and $<$). The ordering on elements is extended to orderings between an element and a composite object (denoted $\preceq, \prec$) and

between composite objects (denoted $\ll$), by requiring that all elements of the composite object observe the ordering relation[7].

Finite **multisets** are composite objects which may contain the same elements several times, that is each element has a certain *multiplicity*. $\emptyset$ denotes the empty multiset, $\{\{a\}\}$ denotes the multiset having only the element a with multiplicity 1. The union (additive) is denoted by $\uplus$: multiplicity is the sum of multiplicities — like in [16]. Union is commutative and associative with unit $\emptyset$, these properties are used implicitly by the prover. We use $\mathcal{M}$ for the multiset of elements of a tree. When two trees have the same elements (that is, their multisets are equal), we call them *equivalent*.

A finite binary **tree** is either ε (empty) or a triplet $\langle L, a, R\rangle$, where L and R are trees. The multiset of a tree has the following property:

Property 1. $\underset{a,L,R}{\forall}\left(\begin{array}{c}\mathcal{M}[\varepsilon] = \emptyset \\ \mathcal{M}[\langle L, a, R\rangle] = \mathcal{M}[L] \uplus \{\{a\}\} \uplus \mathcal{M}[R]\end{array}\right)$

Sorted trees are defined in the following way:

Definition 1.

$$\underset{a,L,R}{\forall}\left(\begin{array}{c}\mathit{IsSorted}[\varepsilon] \\ \mathit{IsSorted}[\langle L, a, R\rangle] \Longleftrightarrow \mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge L \preceq a \preceq R\end{array}\right)$$

Problem and Approach Given two trees X, Y, *merge* them into a tree Z. Moreover, if X, Y are sorted, then Z should be also sorted. The synthesis conjecture has the general structure $\underset{X,Y}{\forall}(I[X,Y] \implies \underset{Z}{\exists} O[X,Y,Z])$, where I is the input condition and O is the output condition.

In the general case X, Y, Z are not required to be sorted, there is no input condition[8] and the output condition $O[X,Y,Z]$ is $(\mathcal{M}[Z] = \mathcal{M}[X] \uplus \mathcal{M}[Y])$, thus we have:

Conjecture 1. $\underset{XY}{\forall\forall}\underset{Z}{\exists}\Big(\mathcal{M}[Z] = \mathcal{M}[X] \uplus \mathcal{M}[Y]\Big)$

For sorted trees we also consider the input condition $I[X,Y]$: $(\mathit{IsSorted}[X] \wedge \mathit{IsSorted}[Y])$ and we add $\mathit{IsSorted}[Z]$ to the output condition, thus we have:

Conjecture 2.

$$\underset{XY}{\forall\forall}(\mathit{IsSorted}[X] \wedge \mathit{IsSorted}[Y]) \Longrightarrow \underset{Z}{\exists}\Big(\mathcal{M}[Z] = \mathcal{M}[X] \uplus \mathcal{M}[Y] \wedge \mathit{IsSorted}[Z]\Big)$$

One may try to prove the conjectures by various induction principles, using one argument or both. For space reasons we focus in the present case study on *domain definition based induction* and on *induction on first argument*: $\underset{X}{\forall} P[X]$ is proven by the induction principle established by the inductive definition of

[7] Note that this introduces exceptions to antisymmetry and transitivity when the empty composite object is involved

[8] This means that the input condition is the logical constant *True* and the implication from the synthesis conjecture reduces to $O[X,Y,Z]$.

the domain. When necessary we refine this induction to a *dynamic induction* method which is applicable to any Noetherian domain: in the induction step we start to prove the induction conclusion $P[t]$ (t ground term) by assuming some induction hypotheses $P[X_0], \ldots, P[X_n]$ according to the inductive definition of the domain ($X_0, \ldots, X_n$ are Skolem constants). If during the proof we need some assumption $P[t']$ where t' (also ground term) represents an object which is strictly smaller than the object represented by t in the Noetherian ordering, then we may assume $P[t']$ holds, that is we can add it to the induction hypotheses. The soundness of this technique is presented in detail in [15], and it allows to discover concrete induction principles based on the general Noetherian induction. The principle of well–founded induction is described as a deduction rule in [18]. Similarly, we use the Noetherian ordering induced by the strict inclusion of the corresponding multisets, which conveniently extends to a meta–ordering between terms, induced by the strict inclusion of the constants occurring in the respective terms. The practical technique for this dynamic induction is described as proof strategy **ST-6** and is illustrated on several examples below.

Moreover we use the *cascading* method pioneered in [3]: when the proof fails, from the failed goal the prover constructs a conjecture synthesis statement which can be used to obtain the auxiliary function which is necessary for the current synthesis. We have been using this for the case of lists in [8, 10], and in this paper we describe it in a more systematic manner as proof strategy **ST-7** and we illustrate it on several examples: all insertion algorithms are generated by cascading starting from failed merging–synthesis proofs. [23] presents a method as a generalization of [18] (an "eureka step" is presented as a rule) for generation auxiliary procedures. Their method seems to be similar to *cascading*, but they use different deductive steps to generate the new statement to be proven and the development of the corresponding auxiliary functions. Moreover, in this paper we present the *cascading* method as an automatic proof technique in the *Theorema* system.

Induction Principle for Binary Trees. We use the induction principle established by the domain definition. In order to prove $\underset{X}{\forall} P[X]$ (*base case*) prove $P[\varepsilon]$; (*induction step*) for Skolem constants a, L_0, R_0 assume *induction hypoteses* $P[L_0], P[R_0]$ and prove *induction conclusion* $P[\langle L_0, a, R_0\rangle]$, where $\langle L_0, a, R_0\rangle$ is the *subject* of the induction conclusion.

In order to synthesize a merging algorithm as a function $F[X, S]$ we prove *Conjecture 1* (take S for Y and T for Z) by transforming S into a Skolem constant S_0 and performing induction on X:

Base case: We prove $\underset{T}{\exists} O[\varepsilon, S_0, T]$. If the proof succeeds to find for T a ground witness $\Im_1[S_0]$ then we know that $F[\varepsilon, S] = \Im_1[S]$.

Step case: For arbitrary but fixed a, L_0 and R_0 (new constants), assume: $\underset{T}{\exists} O[L_0, S_0, T]$ and $\underset{T}{\exists} O[R_0, S_0, T]$, which are Skolemized by introducing two new constants T_1 and T_2. We prove: $\underset{T}{\exists} O[\langle L_0, a, R_0\rangle, S_0, T]$. If the proof succeeds to find a witness $\Im_2[a, L_0, R_0, S_0, T_1, T_2]$, then we know that $F[\langle L, a, R\rangle, S] =$

$\Im_2[a, L, R, S, F[L, S], F[R, S]]$. T_1 and T_2 are replaced by $F[L, S]$ and $F[R, S]$, respectively. Multiple witnesses generate several conditional equalities. Additional arguments to $\Im_2$ may be introduced by dynamic induction as described above and also below at strategy **ST-6**.

In the case of sorted trees, the proof schema is the same, only the given trees $(L_0, R_0, S_0, T_1, T_2)$ are assumed to be sorted, and the witness obtained has to be also sorted.

2.2 Special Inference Rules and Strategies

Following natural style proving, we use *Skolem constants* (denoted with numerical underscore like V_1) introduced for existential assumptions and universal goals, as well as *metavariables* (denoted with star power like T^*) introduced for existential goals. The prover uses classical inference rules (split ground conjunctions, rewrite by equality, etc.) as well as special rules appropriate for trees and multisets.

The strategies are similar to the ones in [8, 15]. The first four strategies are briefly described in [9] and the last three strategies extend the ones in [10] on binary trees. The inference rules: **IR-1**, **IR-2**, **IR-3**, **IR-4**, **IR-5**, **IR-6**, and **IR-8** are adapted for these current case studies of synthesis (they extend the inference rules for lists in [9] and [10]) and all the others presented in this section are novel.

These inference rules and strategies are not specific to the problem of tree merging, but are developed in general for the automation of proof based synthesis of algorithms on lists and trees.

Special Inference Rules.

Each rule is illustrated with an example from the experiments presented in Section 3.

IR-1: *Eliminate assumed formulae from goal.* In a conjunctive goal, delete the part which is already an assumption, or an instance of it. For example goal (34) becomes (35).

IR-2: *Rewrite by equality.* Example: goal (5) is transformed into (6).

IR-3: *Transform to multiple atoms.* This rule transforms parts of the goals or of the assumptions (like e.g. *IsSorted*) into simpler atoms (e.g. by definition). Example: goal (33) becomes (34).

IR-4: *Transform union of M in goal.* Example: goal (7) becomes (8).

IR-5: *Solve metavariables.* Example: goal (32) to (33).

IR-6: *Reduce the goal using assumptions.* Example: transforms goal (35) using the assumption (24) into (36).

IR-7: *Generate branches for trees.* This rule extracts the symbols from a multiset, arranges the symbols and generates branches with new goals. Example: when the goal is $\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \{\{a\}\} \uplus \mathcal{M}[R_0] \uplus \mathcal{M}[S_0]$ extracts the symbols: L_0, a, R_0, S_0 and generates the permutations of (L_0, R_0, S_0). The element a is

considered to be the root of the obtained trees. From all permutations only those are considered which correspond to the current assumptions about ordering.

IR-7-a: *Generate branches for binary non-sorted trees.* Example: if the assumptions are (3) and (4), then the prover generates an OR node with four branches, having goals: (7), (9), (11), and (13).

IR-7-b: *Generate branches for binary sorted trees.* Example: the assumptions are the ones above in **IR-7-a** and also (17), (18), and the goal (19), then the prover generates an OR node with two branches, having in the goal $IsSorted[T^*]$ and also: on one branch goal (20) and on another branch goal (23).

IR-8: *Simple goal conditional assumption.* When the proof fails and the current goal is ground and contains only simple elements (not composite objects), then the proof stops and its result is considered to be this goal (as opposite to *True* when the proof succeeds, or *False* when it fails). Typically this happens in branches generated by the rule **IR-7**, and the unproved goal will become a condition in the synthesized algorithm, as explained below at strategy **ST-4**. Example: goal (37).

Strategies

ST-1: *Quantifier reduction.* This strategy organizes the inference rules for quantifiers (e. g. when applying an induction principle), and it is more effective on goals. For the soundness of the prover it is necessary to keep track of the order in which Skolem constants and metavariables have been introduced, because a Skolem constant which cannot be generated before a certain metavariable cannot be used in a solution for that meta–variable.

ST-2: *Priority of local assumptions.* We consider as local assumptions ground formulae which are generated during the current proof and as global assumptions definitions and properties in the knowledge base. The strategy consists in using first the local assumptions. Example: when the goal is $\mathcal{M}[W^*] = \{\{a\}\} \uplus \mathcal{M}[U_0] \uplus \mathcal{M}[V_0]$ and the assumption is $\mathcal{M}[W_1] = \mathcal{M}[U_0] \uplus \mathcal{M}[V_0]$, the new goal will be $\mathcal{M}[W^*] = \{\{a\}\} \uplus \mathcal{M}[W_1]$ because we give priority to terms containing the Skolem constants generated by the induction hypothesis (they correspond to recursive calls).

ST-3: *Generate more local assumptions.* Example: apply Modus Ponens on local assumptions.

ST-4: *Conditional branches.* Alternative branches generated by the rule **IR-7** may finish with success (proof value is *True*), failure (*False*), or some "simple" goal (proof value is this goal) as explained at **IR-8**. One may see the corresponding OR node of the proof as constituting the logical operation "or" applied to the proof values. If the result is *True* – that is, the disjunction is a logical consequence of the current theory (we can just say "it holds"), then the proof can be considered successful, and in fact it can be transformed by eliminating the false proof values, and by considering the remaining disjunction as a basis for proof by cases – which will now be an AND node, having on each branch the previous proof value as assumption. This approach in fact discovers automatically the basis for the case distinction proof. Moreover, if there are subsets of the disjunction which already hold disjunctively (we can say they are "covering"),

then each such subset can be a basis for the case distinction, thus we can have several successful proof alternatives.

The strategy we employ does not actually transform the proof, because we are only interested in the algorithm. Instead, the respective proof values (simple failed goals) on the branches are taken as conditions for the logical equalities which compose the synthesized algorithm.

ST-5: *Pair multisets.* Often the goal contains an equality like $\mathcal{M}[Y^*] = \mathcal{M}[t_1] \uplus \mathcal{M}[t_2] \uplus \ldots$, where Y^* is the metavariable we need to solve, and $t_1, t_2, \ldots$ are ground terms. The main flow of the proof consists in transforming the union on the LHS of the equality into a single $\mathcal{M}[t]$, because this gives the solution $Y^* \rightarrow t$. Therefore the prover groups pairs of operands of $\uplus$ together (no matter whether they are contingent or not, because commutativity), creating alternatives for different groupings. (Consequently the pair will be transformed into an single multiset term by equality rewriting, or it will be treated by strategy **ST-6** or **ST-7**).

ST-6: *Dynamic Induction.* As mentioned in Subsection 2.1, we use Noetherian induction based on the well–founded ordering between composite objects determined by the strict inclusion of the corresponding multisets. This is checked syntactically by the meta-relation between terms induced by the strict inclusion of the multisets of constants occuring in the terms. When a ground term t' occurring in the goal is smaller than the subject t of the current induction conclusion $P[t]$, then $P[t']$ is used as: $\underset{Y}{\forall}(I[t', Y] \implies \underset{Z}{\exists} O[t', Y, Z])$. Then the prover chooses a ground instantiation s (also part of the goal) for Y, it checks whether $I[t', s]$ holds, it creates a new Skolem constant like for instance Z_1 and it assumes $O[t', s, Z_1]$ holds. In the synthesized algorithm Z_1 will be replaced by $F[t', s]$ (where F is the name of the currently synthesized function). Typically the terms t' and s come from a pair of multiset terms by application of the strategy **ST-5**. Example: goals (40) and (42) are obtained using $P[X]$ (15).

ST-7: *Cascading.* When a pair of multiset terms $t_1[x], t_2[y]$ (x, y constants) is chosen by applying strategy **ST-5**, it may be that there exists no equalities among the current assumptions for reducing it to a single multiset term, or the reduction does not lead to a successful proof. In this case the prover constructs the conjecture: $\underset{X,Y}{\forall}(I[X, Y] \implies \underset{Z}{\exists}(\mathcal{M}[Z] = t_1[X] \uplus t_2[Y] \wedge Q[X, Y, Z]))$, whose proof results in the synthesis of a new function $F[X, Y]$ having the properties required by the current proof situation: $I[X, Y]$ is composed conjunctively from the assumptions which contain *only* the constants x, y (which are replaced by X, Y) and $Q[X, Y, Z]$ is inferred from the current goal.

3 Experiments

3.1 Synthesis of merging on non–sorted binary trees

The proof of *Conjecture 1* by **Induction** on X proceeds as described in Subsection 2.1 for the formula $P[X] : \underset{S}{\forall}\, \underset{T}{\exists}\Big(\mathcal{M}[T] = \mathcal{M}[X] \uplus \mathcal{M}[S]\Big)$. On both branches

(base case and induction step), the universal S is Skolemized to S_0 ("arbitrary but fixed") and the existential T is replaced by the metavariable T^* (unknown witness), according to **ST-1**.

Proof. Base case: Prove

$$\mathcal{M}[T^*] = \mathcal{M}[\varepsilon] \uplus \mathcal{M}[S_0]. \tag{1}$$

Apply **IR-4** using **Property 1** and the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[S_0]. \tag{2}$$

Apply **IR-5**, the obtained substitution is $\{T^* \to S_0\}$.
Induction step: Assume

$$\mathcal{M}[T_1] = \mathcal{M}[L_0] \uplus \mathcal{M}[S_0], \tag{3}$$

$$\mathcal{M}[T_2] = \mathcal{M}[R_0] \uplus \mathcal{M}[S_0] \tag{4}$$

and prove:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle L_0, a, R_0\rangle] \uplus \mathcal{M}[S_0]. \tag{5}$$

Apply **IR-2** using **Property 1** and the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \{\{a\}\} \uplus \mathcal{M}[R_0] \uplus \mathcal{M}[S_0]. \tag{6}$$

Apply **IR-7-a:** using the assumptions (3), (4) and generate and OR node with four branches:
Branch-1: The new goal is:

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \{\{a\}\} \uplus \mathcal{M}[T_2]. \tag{7}$$

Apply **IR-4** and the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle L_0, a, T_2\rangle]. \tag{8}$$

Apply **IR-5** and the obtained substitution on this branch is $\{T^* \to \langle L_0, a, T_2\rangle\}$.
Branch-2: The new goal is:

$$\mathcal{M}[T^*] = \mathcal{M}[T_1] \uplus \{\{a\}\} \uplus \mathcal{M}[R_0]. \tag{9}$$

Apply **IR-4** and the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle T_1, a, R_0\rangle] \tag{10}$$

and the substitution is $\{T^* \to \langle T_1, a, R_0\rangle\}$.
Branch-3: The new goal is:

$$\mathcal{M}[T^*] = \mathcal{M}[R_0] \uplus \{\{a\}\} \uplus \mathcal{M}[T_1]. \tag{11}$$

Apply **IR-4**, the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle R_0, a, T_1\rangle] \tag{12}$$

and the substitution is $\{T^* \to \langle R_0, a, T_1\rangle\}$.
Branch-4: The new goal is:

$$\mathcal{M}[T^*] = \mathcal{M}[T_2] \uplus \{\{a\}\} \uplus \mathcal{M}[L_0]. \tag{13}$$

Apply **IR-4**, the goal becomes:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle T_2, a, L_0\rangle] \tag{14}$$

and the substitution is $\{T^* \to \langle T_2, a, L_0\rangle\}$.

Since all branches succeed, each of them generates an alternative algorithm, thus we have:

Algorithm 1 Concatenation of trees.

$$\underset{a,L,R,S}{\forall} \left(\begin{array}{c} \mathit{Conc}[\varepsilon, S] = S \\ \mathit{Conc}[\langle L, a, R\rangle, S] = \langle L, a, \mathit{Conc}[R, S]\rangle \end{array} \right)$$

as well as three other concatenation algorithms where the RHS of the second equality is: $\langle F[L, S], a, R\rangle$, $\langle R, a, F[L, S]\rangle$, or $\langle F[R, S], a, L\rangle$.

3.2 Synthesis of merging on sorted binary trees

The proof of *Conjecture 2* by **Induction** on X proceeds as described in Subsection 2.1 for the formula $P[X]$:

$$\underset{S}{\forall}\Big((\mathit{IsSorted}[X] \wedge \mathit{IsSorted}[S]) \Longrightarrow \underset{T}{\exists}(\mathcal{M}[T] = \mathcal{M}[X] \uplus \mathcal{M}[S] \wedge \mathit{IsSorted}[T]\Big) \tag{15}$$

On both branches (base case and induction step), the universal S is Skolemized to S_0 ("arbitrary but fixed") and the existential T is replaced by the metavariable T^* (unknown witness), according to **ST-1**. The proof is similar with the previous one, with the difference that at the induction step in addition to the induction hypothesis (3), (4) one obtains more assumptions regarding the ordering.

Proof.

$$\mathit{IsSorted}[T_1] \wedge \mathit{IsSorted}[T_2], \tag{16}$$

$$\mathit{IsSorted}[\langle L_0, a, R_0\rangle]. \tag{17}$$

By **IR-3** from (17) obtain:

$$\mathit{IsSorted}[L_0] \wedge L_0 \preceq a \wedge a \preceq R_0 \wedge \mathit{IsSorted}[R_0]. \tag{18}$$

The goal is similar to (6), in addition T^* has to be sorted:

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \{\{a\}\} \uplus \mathcal{M}[R_0] \uplus \mathcal{M}[S_0] \wedge \mathit{IsSorted}[T^*]. \tag{19}$$

Apply **IR-7-b** using the induction hypothesis (3), (4), and also (17), (18) and generate two branches:
Branch-1: The new goal is

$$\mathcal{M}[T^*] = \mathcal{M}[T_1] \uplus \{\{a\}\} \uplus \mathcal{M}[R_0] \wedge \mathit{IsSorted}[T^*]. \tag{20}$$

Apply **IR-4** and the goal is:

$$\mathcal{M}[T^*] = \mathcal{M}[\langle T_1, a, R_0\rangle] \wedge \mathit{IsSorted}[T^*]. \tag{21}$$

Apply **IR-5**, the obtained substitution is $\{T^* \to \langle T_1, a, R_0\rangle\}$ and the remaining goal is:

$$\mathit{IsSorted}[\langle T_1, a, R_0\rangle]. \tag{22}$$

Apply **IR-3** using **Property 1**, **IR-2** using (16), (18), **IR-6** using (3), (18) and the remaining goal is $S_0 \preceq a$. The proof fails on this branch.
Branch-2: The new goal is

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \{\{a\}\} \uplus \mathcal{M}[T_2] \wedge \mathit{IsSorted}[T^*]. \tag{23}$$

Similarly, the obtained substitution is $\{T^* \to \langle L_0, a, T_2\rangle\}$ and the remaining goal is $a \preceq S_0$. The proof fails.

However synthesis is still possible by the technique described below.

Cascading–synthesis of insertion on binary trees: The prover applies strategy **ST-5** (pair multisets) by grouping $\{\{a\}\}$ and $\mathcal{M}[R_0]$ – for which we already know $a \preceq R_0$ – and then strategy **ST-7** (cascading), producing the conjecture:

Conjecture 3.
$$\underset{a,R}{\forall}\Big(\big(\mathit{IsSorted}[R] \wedge a \preceq R\big) \Longrightarrow \underset{S}{\exists}\Big(\mathcal{M}[S] = \{\{a\}\} \uplus \mathcal{M}[R] \wedge \mathit{IsSorted}[S]\Big)\Big)$$

By proving this conjecture we obtain the algorithm *Prepend* which places a given element as the leftmost node of a given tree:

Algorithm 2 Prepend an element to a tree.
$$\underset{a,b,L,R}{\forall}\begin{pmatrix} \mathit{Prepend}[a, \varepsilon] = \langle \varepsilon, a, \varepsilon\rangle \\ \mathit{Prepend}[a, \langle L, b, R\rangle] = \langle \mathit{Prepend}[a, L], b, R\rangle \end{pmatrix}$$

However, by using this auxiliary function the main proof still does not succeed, therefore a merging algorithm cannot be found.

Similarly, for the goal (23), by grouping $\mathcal{M}[L_0]$ and $\{\{a\}\}$ – for which we already know $L_0 \preceq a$, we obtain the conjecture for the synthesis of the auxiliary function *Append*, which places a given element at the rightmost node of a given tree, but in this case the synthesis of the merging algorithm still fails.

If for proving (19) we group $\mathcal{M}[S_0]$ and $\{\{a\}\}$, then there is no more ordering between them, and the conjecture is:

Conjecture 4. $\underset{a,X}{\forall}\Big(\mathit{IsSorted}[X] \Longrightarrow \underset{S}{\exists}\Big(\mathcal{M}[S] = \{\{a\}\} \uplus \mathcal{M}[X] \wedge \mathit{IsSorted}[S]\Big)\Big)$

By proving this conjecture we obtain the function *Insert* which places a given element as the appropriate position in a sorted tree.

Prove *Conjecture 4* by applying **Induction** on X.

Proof. Base case: The obtained substitution is $\{T^* \to \langle \varepsilon, a, \varepsilon \rangle\}$.
Induction step: Assume

$$\mathcal{M}[S_1] = \{\{a\}\} \uplus \mathcal{M}[L_0], \tag{24}$$

$$\mathcal{M}[S_2] = \{\{a\}\} \uplus \mathcal{M}[R_0], \tag{25}$$

$$\mathit{IsSorted}[S_1] \wedge \mathit{IsSorted}[S_2], \tag{26}$$

$$\mathit{IsSorted}[\langle L_0, b, R_0 \rangle], \tag{27}$$

$$\mathit{IsSorted}[L_0] \wedge L_0 \preceq b \wedge b \preceq R_0 \wedge \mathit{IsSorted}[R_0] \tag{28}$$

and prove:

$$\mathcal{M}[S^*] = \{\{a\}\} \uplus \mathcal{M}[\langle L_0, b, R_0 \rangle] \wedge \mathit{IsSorted}[S^*]. \tag{29}$$

Apply **IR-2** using **Property 1** and the new goal is:

$$\mathcal{M}[S^*] = \{\{a\}\} \uplus \mathcal{M}[L_0] \uplus \{\{b\}\} \uplus \mathcal{M}[R_0] \wedge \mathit{IsSorted}[S^*]. \tag{30}$$

Apply **IR-7-b** considering b to be the root of the obtained tree, using the assumptions (24), (25), (27) and generate two branches:
Branch-1: The new goal is:

$$\mathcal{M}[S^*] = \mathcal{M}[S_1] \uplus \{\{b\}\} \uplus \mathcal{M}[R_0] \wedge \mathit{IsSorted}[S^*]. \tag{31}$$

Apply **IR-4** using **Property 1** and prove:

$$\mathcal{M}[S^*] = \mathcal{M}[\langle S_1, b, R_0 \rangle] \wedge \mathit{IsSorted}[S^*]. \tag{32}$$

Apply **IR-5**, the substitution is $\{T^* \to \langle S_1, b, R_0 \rangle\}$ and the new goal is:

$$\mathit{IsSorted}[\langle S_1, b, R_0 \rangle]. \tag{33}$$

Apply **IR-3** using **Definition 1** and the goal becomes:

$$\mathit{IsSorted}[S_1] \wedge S_1 \preceq b \wedge b \preceq R_0 \wedge \mathit{IsSorted}[R_0]. \tag{34}$$

Apply $\mathbf{IR-1}$ using (26), (28) and the remaining goal is: $S_1 \preceq b$. (35)

Apply $\mathbf{IR-6}$ using (24) and the new goal is: $a \leq b \wedge L_0 \preceq b$. (36)

Apply $\mathbf{IR-1}$ using (28) and the remaining goal is: $a \leq b$. (37)

By **IR-8**, (37) becomes the conditional assumption on this branch.
Branch-2: The new goal is:

$$\mathcal{M}[S^*] = \mathcal{M}[L_0] \uplus \{\{b\}\} \uplus \mathcal{M}[S_2] \wedge \mathit{IsSorted}[S^*]. \tag{38}$$

Similar as in the previous branch, the obtained substitution is $\{T^* \to \langle L_0, b, S_2 \rangle\}$ and the conditional assumption on this branch is $b \leq a$. By **ST-4** we obtain:

Algorithm 3 Insertion in a sorted tree.

$$\underset{a,b,L,R}{\forall}\left(\begin{array}{c} Ins[a,\varepsilon] = \langle \varepsilon, a, \varepsilon\rangle \\ Ins[a,\langle L,b,R\rangle] = \begin{cases} \langle Ins[a,L],b,R\rangle, \textit{ if } a \leq b \\ \langle L,b,Ins[a,R]\rangle, \textit{ if } b < a \end{cases} \end{array}\right)$$

By the cascading strategy **ST-7**, we continue the proof of the merging conjecture by replacing in the goal (19) the subterm $\{\{a\}\} \uplus \mathcal{M}[S_0]$ (which generated the conjecture for synthesizing *Ins*) by the corresponding instance $Ins[a, S_0]$:

Proof.

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \mathcal{M}[R_0] \uplus \mathcal{M}[Ins[a, S_0]] \wedge \mathit{IsSorted}[T^*]. \tag{39}$$

Apply strategy **ST-5** (pair multisets) and **ST-6** (dynamic induction) to $\mathcal{M}[R_0]$ and $\mathcal{M}[Ins[a, S_0]]$. The object represented by R_0 is smaller in the well founded ordering than the object $\langle R_0, a, L_0\rangle$, which is the subject of the current induction conclusion (formula (15) with substitution $X \longrightarrow \langle R_0, a, L_0\rangle$). Therefore we may assume $P[R_0]$ holds, and use $Ins[a, S_0]$ for the instantiation of the second argument, thus by Skolemization we obtain an object R_1 observing:

$$\mathcal{M}[R_1] = \mathcal{M}[R_0] \uplus \mathcal{M}[Ins[a, S_0]] \wedge \mathit{IsSorted}[R_1]. \tag{40}$$

Apply equality rewriting using this to transform goal (39) into:

$$\mathcal{M}[T^*] = \mathcal{M}[L_0] \uplus \mathcal{M}[R_1] \wedge \mathit{IsSorted}[T^*]. \tag{41}$$

Since the object represented by L_0 is smaller in the well founded ordering than $\langle R_0, a, L_0\rangle$, we can again apply Noetherian induction to obtain L_1 with:

$$\mathcal{M}[L_1] = \mathcal{M}[L_0] \uplus \mathcal{M}[R_0] \wedge \mathit{IsSorted}[L_1]. \tag{42}$$

Apply equality rewriting using this to transform goal (41) into:

$$\mathcal{M}[T^*] = \mathcal{M}[L_1] \wedge \mathit{IsSorted}[T^*] \tag{43}$$

which gives the solution $T^* = L_1$ and the proof succeeds, giving the algorithm:

Algorithm 4 Merge sorted trees, version 1.

$$\underset{a,L,R,S}{\forall}\left(\begin{array}{c} Merge[\varepsilon, S] = S \\ Merge[\langle L,a,R\rangle, S] = Merge[L, Merge[R, Ins[a, S]]] \end{array}\right)$$

Note how a *nested recursion* — for which a concrete induction principle would be difficult to guess — is produced automatically by our method. This algorithm is interesting because it is probably optimal: essentially it inserts one by one the elements of the first tree into the (sorted) second tree. Note also that the assumptions (17) and (18) are not necessary for the success of the proof, and indeed the algorithm produces a sorted tree even if the first argument is not sorted. Similarly to the situation with lists [10], since the first argument does

not need to be sorted, both this algorithm and the next one can be used for sorting as $Merge[T, \varepsilon]$. Sorting is performed by traversing the tree and inserting the elements one by one in a new sorted tree, which appears to be optimal.

There are many ways in which the subterms of the RHS of the equality in (19) can be grouped pairwise and then be used in a similar manner to cascade new auxiliary functions and to produce new merging algorithms. We present here only one other alternative, which is interesting because it is tail recursive, and only slightly less efficient than the previous one.

The proof is modified as follows:

Proof. Strategy **ST-5** (pair multisets) on the goal (39) groups the subterms $\mathcal{M}[L_0]$ and $\mathcal{M}[R_0]$, and then strategy **ST-7** (cascade) generates the conjecture:

$$\underset{L,R}{\forall}\ \underset{X}{\exists} \mathcal{M}[X] = \mathcal{M}[L] \uplus \mathcal{M}[R] \wedge \mathit{IsSorted}[X] \tag{44}$$

The proof of this is very similar to the proof of *Conjecture 1* (for synthesis of merging on non-sorted trees) presented at the beginning of Section 3.1, with the difference that the proof starts with the additional assumptions $\mathit{IsSorted}[L_0, a, R_0]$, $\mathit{IsSorted}[T_1]$, $\mathit{IsSorted}[T_2]$, while the goal has also $\mathit{IsSorted}[T^*]$. Therefore the proof succeeds on the first branch with the same witness $\langle L_0, a, T_2 \rangle$, which is proven sorted by applying the definition and the properties of ordering to the assumptions — so the same **Algorithm 1** *Conc* also concatenates sorted trees into a sorted tree.

Strategy **ST-7** (cascading) replaces in goal (19) the pair $\mathcal{M}[L_0] \uplus \mathcal{M}[R_0]$ by $\mathcal{M}[\mathit{Conc}[L_0, R_0]]$ to get:

$$\mathcal{M}[T^*] = \mathcal{M}[\mathit{Conc}[L_0, R_0]] \uplus \mathcal{M}[\mathit{Ins}[a, S_0]] \wedge \mathit{IsSorted}[T^*]. \tag{45}$$

Since $\mathit{Conc}[L_0, R_0]$ is smaller in the well–founded ordering than $\langle L_0, a, R_0 \rangle$, strategy **ST-6** (dynamic induction) uses it together with the instantiation $\mathcal{M}[\mathit{Ins}[a, S_0]]$ for the second argument, and obtains L_2 with the property:

$$\mathcal{M}[L_2] = \mathcal{M}[R_0] \uplus \mathcal{M}[\mathit{Ins}[a, S_0]] \wedge \mathit{IsSorted}[L_2]. \tag{46}$$

By equality rewriting this transforms the goal (45) into:

$$\mathcal{M}[T^*] = \mathcal{M}[L_2] \wedge \mathit{IsSorted}[T^*] \tag{47}$$

which gives the solution $T^* = L_2$ and the proof succeeds, giving the algorithm:

Algorithm 5 Merge sorted trees, version 2.

$$\underset{a,L,R,S}{\forall} \left(\begin{array}{c} \mathit{Merge}[\varepsilon, S] = S \\ \mathit{Merge}[\langle L, a, R \rangle, S] = \mathit{Merge}[\mathit{Conc}[L, R], \mathit{Ins}[a, S]]] \end{array} \right)$$

Similarly to the other version, since the first argument does not need to be sorted, this can also be used for sorting as $\mathit{Merge}[T, \varepsilon]$. This algorithm is interesting because it is tail recursive, even as it is slightly less efficient than the previous one.

4 Conclusions and Further Work

Our experiments demonstrate the possibility of automatic synthesis of complex algorithms on (possibly sorted) binary trees, using the notion of multiset. In certain cases, depending on the proof strategy, several algorithms are produced for the same function or from different proofs the same algorithm is produced.

Even as some of the synthesized algorithms are relatively straightforward and sometimes not optimal, this case study helps in at least three ways. First, the study develops the underlying theory and helps understand better the principles of theory exploration, for instance by a parallel development one has hints about interesting functions on trees suggested by the classical operations on multisets (insertion corresponds to union with one element, merging corresponds to union, etc.). Second, the study helps to develop efficient proof methods for these domains, in particular by using specific inference rules and strategies which are also taylored for synthesis proofs, notably for discovering induction principles for nested recursion. Finally, the various algorithms which are produced can constitute a test field for methods of automatic evaluation of efficiency, time and space consumption, etc.

A distinctive feature of our approach is the use of natural–style proofs, which is facilitated by the *Theorema* system. The natural style of proving (as formula notation, as proof text, and as inference steps) has the advantage of allowing human inspection in an intuitive way, and this facilitates the development of intuitive inference rules which embed the knowledge about the underlying domains.

The experiments presented here continue our previous work on synthesis of deletion algorithms and sorting algorithms on lists using multisets and is prerequisite for further work on synthesis of more complex algorithms for sorting and searching, including algorithms which combine operations on several domains.

References

1. Blizard, W.D.: Multiset Theory. Notre Dame Journal of Formal Logic **30**(1), 36–66 (1989). https://doi.org/10.1305/ndjfl/1093634995
2. Buchberger, B.: Theory Exploration with Theorema. Analele Universitatii Din Timisoara, Seria Matematica-Informatica **XXXVIII**(2), 9–32 (2000)
3. Buchberger, B.: Algorithm Invention and Verification by Lazy Thinking. Analele Universitatii din Timisoara, Seria Matematica - Informatica **XLI**, 41–70 (2003)
4. Buchberger, B., Craciun, A.: Algorithm Synthesis by Lazy Thinking: Using Problem Schemes. In: Proceedings of SYNASC 2004. pp. 90–106 (2004)
5. Buchberger, B., Dupre, C., Jebelean, T., Kriftner, F., Nakagawa, K., Vasaru, D., Windsteiger, W.: The Theorema project: A progress report. In: Calculemus 2000. pp. 98–113. A.K. Peters, Natick, Massachusetts (2000)
6. Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-Assisted Natural-Style Mathematics. Journal of Formalized Reasoning **9**(1), 149–185 (2016). https://doi.org/10.6092/issn.1972-5787/4568

7. Bundy, A., Dixon, L., Gow, J., Fleuriot, J.: Constructing Induction Rules for Deductive Synthesis Proofs. Electronic Notes Theoretical Computer Science **153**, 3–21 (March 2006). https://doi.org/10.1016/j.entcs.2005.08.003
8. Dramnesc, I., Jebelean, T.: Synthesis of List Algorithms by Mechanical Proving. Journal of Symbolic Computation **68**, 61–92 (2015). https://doi.org/10.1016/j.jsc.2014.09.030
9. Dramnesc, I., Jebelean, T.: Case Studies on Algorithm Discovery from Proofs: The *Delete* Function on Lists and Binary Trees using Multisets. In: SISY 2019. pp. 213–220. IEEE Xplore (2019)
10. Dramnesc, I., Jebelean, T.: Proof–Based Synthesis of Sorting Algorithms Using Multisets in *Theorema*. In: FROM 2019. pp. 76–91. EPTCS 303 (2019). https://doi.org/10.4204/EPTCS.303.6
11. Dramnesc, I., Jebelean, T., Stratulat, S.: Combinatorial Techniques for Proof-based Synthesis of Sorting Algorithms. In: SYNASC 2015. pp. 137–144 (2015). https://doi.org/10.1109/SYNASC.2015.30
12. Dramnesc, I., Jebelean, T., Stratulat, S.: Theory Exploration of Binary Trees. In: SISY 2015. pp. 139 – 144. IEEE (2015). https://doi.org/10.1109/SISY.2015.7325367
13. Dramnesc, I., Jebelean, T., Stratulat, S.: A Case Study on Algorithm Discovery from Proofs: The Insert function on Binary Trees. In: SACI 2016. pp. 231–236. IEEE (2016). https://doi.org/10.1109/SACI.2016.7507376
14. Dramnesc, I., Jebelean, T., Stratulat, S.: Proof-based Synthesis of Sorting Algorithms for Trees. In: LATA 2016. pp. 562–575. Springer (2016). https://doi.org/10.1007/978-3-319-30000-9_43
15. Dramnesc, I., Jebelean, T., Stratulat, S.: Mechanical Synthesis of Sorting Algorithms for Binary Trees by Logic and Combinatorial Techniques. Journal of Symbolic Computation **90**, 3–41 (2019). https://doi.org/10.1016/j.jsc.2018.04.002
16. Knuth, D.E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, 3 edn. (1998). https://doi.org/10.1137/1012065
17. Korukhova, Y.: Automatic Deductive Synthesis of Lisp Programs in the System ALISA. In: JELIA 2006. pp. 242–252. Springer LNAI 4160 (2006)
18. Manna, Z., Waldinger, R.: A Deductive Approach to Program Synthesis. ACM Transactions on Programming Languages and System **2**(1), 90–121 (1980). https://doi.org/10.1145/357084.357090
19. Manna, Z., Waldinger, R.: The Logical Basis for Computer Programming, vol. 1: Deductive Reasoning. Addison-Wesley (1985). https://doi.org/10.2307/2275898
20. Manna, Z., Waldinger, R.: Fundamentals Of Deductive Program Synthesis. IEEE Transactions on Software Engineering **18**(8), 674–704 (1992). https://doi.org/10.1109/32.153379
21. Radoaca, A.: Properties of Multisets Compared to Sets. In: SYNASC 2015. pp. 187–188 (2015). https://doi.org/10.1109/SYNASC.2015.37
22. Smith, D.R.: Kids: a semiautomatic program development system. IEEE Transactions on Software Engineering **16**(9), 1024–1043 (1990). https://doi.org/10.1109/32.578788
23. Traugott, J.: Deductive Synthesis of Sorting Programs. Journal of Symbolic Computation **7**(6), 533–572 (1989). https://doi.org/10.1016/S0747-7171(89)80040-9
24. Windsteiger, W.: Theorema 2.0: A System for Mathematical Theory Exploration. In: ICMS'2014. LNCS, vol. 8592, pp. 49–52 (2014). https://doi.org/10.1007/978-3-662-44199-2_9