Deductive Synthesis of *Min-Max-Sort* Using Multisets in *Theorema*

Isabela Drămnesc
Department of Computer Science
West University
Timişoara, Romania
Email: isabela.dramnesc@e-uvt.ro

Tudor Jebelean
Research Institute for Symbolic Computation,
Johannes Kepler University,
Linz, Austria
Email: Tudor.Jebelean@jku.at

***Abstract*—We demonstrate the deductive synthesis of the *Min-Max-Sort* algorithm using multisets in the frame of the *Theorema* system. Starting from the logical specification of the sorting function (input and output conditions), we show how to construct a synthesis conjecture, from whose proof the algorithm can be constructed. We choose those inference methods and induction principles such that the algorithm consists of selecting at each step the minimum and the maximum of the list, and moving them at the ends of the list. During the main proof new conjectures are produced for the synthesis of auxiliary algorithms, and this process repeats in a cascading fashion until all necessary algorithms are produced. Our proof techniques, which are in natural style, include a novel approach using multisets and the use of cover sets for realizing Noetherian induction. The later has the advantage that the concrete induction hypotheses are created dynamically during the proof of the corresponding induction conclusion, thus no concrete induction principle or algorithm scheme is needed in advance. The synthesis mechanism is implemented in the frame of the *Theorema* system [2], [24] which allows the construction of mathematical theories, proving in natural style, and computing with the synthesized algorithms.**

***Index Terms*—automated reasoning, algorithm synthesis, lists, multisets, Theorema**

I. Introduction

The problem of automated algorithm synthesis as an alternative to algorithm verification is a very interesting and challenging problem. [21] gives an overview of the most common approaches used to tackle the synthesis problem. For a survey of the synthesis methods see [8]. We use the proof–based synthesis approach which consists in generating a running algorithm satisfying a given specification. The specification consists in an input and an output condition. From this the corresponding logical conjecture is obtained and from the proof of this conjecture the algorithm is extracted.

In this paper we synthesize *Min-Max-Sort* in the *Theorema* system [24]. We extend the techniques in [9], [10] by adding some specific inference rules and strategies. We also transform the synthesized algorithm into a tail recursive one, then we describe the version which uses a flag in order to avoid unnecessary recursions.

A. Related work and originality

In [6] six sorting algorithms are derived by applying transformation rules. This work was extended in [1] by adding another symmetry, see also [13]. In [12] some transformation techniques which complement the techniques in [6] are added and several sorting algorithms are derived. A top–down approach is used in [16] to synthesize a large family of sorting algorithms. A classification of sorting algorithms is designed in [16]. Deductive tableau techniques for recursive algorithm synthesis are introduced in [18] which are applied in [22] to manually synthesize several sorting algorithms. [15] synthesizes automatically several functions in Lisp by applying deductive techniques [18] and some heuristics and rippling [3]. Significant work also has been done in sorting algorithm synthesis by using the proof–based approach on lists [8] and on binary trees [11]. The difference from our current work is that we use multisets [19] in the entire process of algorithm synthesis and we apply different proof techniques.

A brief description of eight iterative sorting algorithms is given in [14]. The authors propose a hardware optimization to be applied in order to speed–up the process of sorting. They also compare the performance of the 8 sorting algorithms with respect to resource usage and time execution. Most of researchers find the performance (space and time) of recursive algorithms as being lower compared to tail recursive and iterative algorithms. In this paper we discover one sorting algorithm from proof by applying proof–based techniques and then we transform the obtained recursive version into its tail recursive, then obtain one improved version which uses a flag.

The first study on transforming recursive functions into more efficient ones [5], including recursion removal [4] uses some transformation rules and strategies. Their implementation based on schemas is in [7]. A method for transforming general recursion to iteration based on incrementalization is given in [17]. We transform recursive algorithms into tail recursive ones, and then we describe their version which use a flag. From this version one can easily obtain their functional and iterative versions.

In [23] five enhanced iterative algorithms are presented (including *Min-Max-Sort*) and their complexity is analyzed. The experiments show that the enhanced versions of algorithms have a better performance and a reduced complexity. In this paper we discover from proof the recursive version of *Min-Max-Sort* together with the auxiliary functions and then we derive more efficient versions, namely the tail recursive and

the one with a flag.

In [10] we synthesize *Max-Sort* and *Bubble-Sort* algorithm and then we transform them into more efficient versions: tail recursive, an improved one which uses a flag, functional and imperative. The current paper is similar with [10], but here we add some more inference rules and strategies, we synthesize *Min-Max-Sort* (an improved *Bubble-Sort*). Then we apply the transformation rules from [10] to obtain more efficient versions of *Min-Max-Sort*.

The main novelty in this paper consists in: (a) the use of *multisets* which allows to express very naturally the fact that two lists have the same elements (no need to use the permutation predicate), (b) the extension of our recent work in [9], [10] by adding some specific proof–techniques (inference rules and strategies) which lead to more efficient proofs, (c) the synthesis of the recursive version of *Min-Max-Sort* (which selects both the minimum and the maximum) and which is an improved version of *Bubble-Sort*and the synthesis of the auxiliary functions, (d) the derivation of improved (tail recursive and the one with a flag) versions of the synthesized *Min-Max-Sort* algorithm by applying the transformation methods from [10] together with their auxiliary functions.

The synthesis prover is implemented in the frame of the *Theorema* system [2], [24] which allows the construction of mathematical theories, proving in natural style, and computing with the synthesized algorithms.

II. Context

A. Notations

As in the *Theorema* system, in this presentation for function and for predicate application we use square brackets (like e. g. $f[x]$ instead of $f(x)$ and $P[a]$ instead of $P(a)$). Quantified variables are placed under the quantifier, as in $\underset{X}{\forall}$ and $\underset{X}{\exists}$.

We use two main types of objects from a totally ordered domain: simple objects (*elements*) and composite objects (*lists* and *multisets*). Simple objects are members of composite objects. The type of the objects is not used explicitly in the proofs, but it is used implicitly based on the notation convention. Lower case letters (e.g. a, b, c) denote constants and variables, and upper case letters denote multisets (e.g. A, B, C), respectively lists (e.g. U, V, W, X, Y, Z). A similar convention applies to function names: starting with lower case is a function whose result is an element, upper case is a function. Most of the basic functions are denoted by special symbols: $\{\!\{a\}\!\}$ is the multiset containing the element a with multiplicity 1, $\uplus$ denotes the additive union of multisets[1], $\frown$ (*append*) adds an element at the end of a list, and $\smile$ (*cons*) adds an element at the beginning of a list.

Some of the basic properties of ordering, of multiset union, list functions, etc. are explicitely used in the proofs, while some are used implicitely by the prover.

Additionally the theory contains the functions *head* and *Tail* (as reverses of $\smile$), as well as *Front* and *last* (as reverses of $\frown$). The relevant properties are:

[1]The multiplicity in the union is the sum of multiplicities

Property 1. $$\underset{a,U}{\forall}\begin{pmatrix} \mathit{head}[a \smile U] = a \\ \mathit{Tail}[a \smile U] = U \\ U = \mathit{head}[U] \smile \mathit{Tail}[U] \end{pmatrix}$$

Property 2. $$\underset{U}{\forall}\begin{pmatrix} \mathit{Front}[U \frown a] = U \\ \mathit{last}[U \frown a] = a \\ U = \mathit{Front}[U] \frown \mathit{last}[U] \end{pmatrix}$$

The multiset of a list observes:

Property 3. $$\underset{a,U}{\forall}\begin{pmatrix} \mathcal{M}[\langle\rangle] = \emptyset \\ \mathcal{M}[a \smile U] = \{\!\{a\}\!\} \uplus \mathcal{M}[U] \end{pmatrix}$$

Sorted lists have the definition:

Definition 1.
$$\underset{a,U}{\forall}\begin{pmatrix} \mathit{IsSorted}[\langle\rangle] \\ \mathit{IsSorted}[U \frown a] \Longleftrightarrow (U \leq a \wedge \mathit{IsSorted}[U]) \end{pmatrix}$$

B. The Approach

We follow the classical approach [20]: starting from the specification consistsing in the input condition and the output condition, one constructs a *synthesis conjecture*, from whose constructive proof the algorithm is extracted.
For univariate functions the specification has the structure $I[X],\ O[X, Y]$ and the conjecture is:

Conjecture 1. $\underset{X}{\forall}(I[X] \implies \underset{Y}{\exists} O[X, Y])$.

For a bivariate function the specification is $I[X, Y]$, $O[X, Y, Z]$, and the conjecture is:

Conjecture 2. $\underset{X,Y}{\forall}(I[X, Y] \implies \underset{Z}{\exists} O[X, Y, Z])$.

In our case the main problem consists in: given a list find its sorted version, thus the input specification is missing. However for the auxiliary functions the specification may have also some specific input condition.

C. Proof Techniques

Some of the classical natural style inference rules which are already implemented in *Theorema* are also used: the deduction rule for implicational goal, the splitting of conjunctional assumption, Skolemization of the universal goal, metavariable for existential goal, etc.

Additionally the special prover uses techniques from which some are appropriate for synthesis, and some are domain specific. The novelty of the approach resides mainly in the techniques which are triggered by the use of multisets.

IR-1: *Forward inference.* Nonclausal unit resolution between assumptions.

IR-2: *Backward inference.* Nonclausal resolution between an unit assumption and the goal.

IR-3: *Reduce composite argument.* Split an atom containing composite arguments into atoms containing simple arguments (variables and constants).

IR-4: *Solve metavariable.* Obtain the value of a metavariable from an equality of multiset terms.

IR-5: *Expand multiset.* Transform a multiset term into several terms.

IR-6: *Compress multiset.* Group two or more multiset terms into one.

IR-7: *Use equivalence.* Use the equivalence (induced by the equality of multisets) for transforming an atom.

IR-8: *Several constants.* Generate cases w.r.t. the ordering of two constants when a second constant element is introduced in the proof. In general when a new constant is introduced in the proof, generate cases w. r. t. the ordering possibilites between this constant and the old ones.

ST-1: *Cover set.* Generate proof alternatives using a cover set for a constant or for a metavariable.

ST-2: *Induction.* Generate an induction hypothesis and a recursive term in the goal for a term which is smaller (with respect to the Noetherian metaordering induced by the strict inclusion of multisets) than the current target constant.

ST-3: *Cascading.* Generate a conjecture for the synthesis of a necessary auxiliary function, the appropriate property and the appropriate term in the current goal.

ST-4: *Pair multisets.* For a pair of multiset terms in the goal, generate an equal term, by using a known property.

ST-5: *Split.* For a multiset term in the goal, generate an equal pair of terms, by using a known property.

ST-6: *Split goal equation.* Use heuristics to split a goal equations which contains several metavariables and/or several multiset terms into equations which can be solved indepenedently.

III. Synthesis of *Min-Max-Sort*

A. Synthesis of the Main Function

The function *MMSort* is generated from the proof of the synthesis conjecture:

Conjecture 3. $\underset{XV}{\forall\exists}(\mathcal{M}[V] = \mathcal{M}[X] \ \wedge \ \mathit{IsSorted}[V])$.

Proof 1:

By classical quantified inference rules, universal X is Skolemized to the *target constant* X_0, producing the *target goal*:

$$\underset{V}{\exists}\mathcal{M}[V] = \mathcal{M}[X_0] \ \wedge \ \mathit{IsSorted}[V] \tag{1}$$

and the existential V becomes the metavariable V^*:

$$\mathcal{M}[V^*] = \mathcal{M}[X_0] \ \wedge \ \mathit{IsSorted}[V^*]. \tag{2}$$

Strategy **ST-1** (cover set) is applied to the metavariable V^* using the cover set: $\{\langle\rangle, a^* \frown \langle\rangle, \ a^* \frown (U^* \frown b^*)\}$:
Case 1.1. $V^* = \langle\rangle$. The goal (2) becomes:

$$\mathcal{M}[\langle\rangle] = \mathcal{M}[X_0] \ \wedge \ \mathit{IsSorted}[\langle\rangle]. \tag{3}$$

By inference rule **IR-2** (backward inference) using **Definition 1** the goal (3) becomes:

$$\mathcal{M}[\langle\rangle] = \mathcal{M}[X_0]. \tag{4}$$

By **ST-1** the proof succeeds on this branch, the witness is $\langle\rangle$, the condition on the input is $X = \langle\rangle$, and the cumulated condition on the input for the next branch is $X_0 \neq \langle\rangle$.

Case 1.2. The goal (2) becomes:

$$\mathcal{M}[a^* \frown \langle\rangle] = \mathcal{M}[X_0] \ \wedge \ \mathit{IsSorted}[a^* \frown \langle\rangle]. \tag{5}$$

By inference rule **IR-2** (backward inference) using **Definition 1** the goal (3) becomes:

$$\mathcal{M}[a^* \frown \langle\rangle] = \mathcal{M}[X_0]. \tag{6}$$

By inference rule **IR-2** (backward inference) using **Property 1** the goal (6) becomes:

$$\mathcal{M}[a^* \frown \langle\rangle] = \mathcal{M}[\mathit{head}[X_0] \frown \mathit{Tail}[X_0]]. \tag{7}$$

By inference rule **IR-5** the goal becomes:

$$\{\!\{a^*\}\!\} \uplus \mathcal{M}[\langle\rangle] = \mathcal{M}[\mathit{head}[X_0] \frown \mathit{Tail}[X_0]]. \tag{8}$$

By strategy **ST-6** and inference rule **IR-4** the proof succeeds on this branch, the witness is $a^* \rightarrow \mathit{head}[X_0]$, the condition on the input is $\mathit{Tail}[X_0] = \langle\rangle$, and the cumulated condition on the input for the next branch is $X_0 \neq \langle\rangle \ \wedge \ \mathit{Tail}[X_0] \neq \langle\rangle$.
Case 1.3 $V^* = a^* \frown (U^* \frown b^*)$. The conditions on X_0 from the previous branch are added as assumptions:

$$X_0 \neq \langle\rangle. \tag{9}$$

$$\mathit{Tail}[X_0] \neq \langle\rangle. \tag{10}$$

The goal (2) becomes:

$$\mathcal{M}[a^* \frown (U^* \frown b^*)] = \mathcal{M}[X_0] \ \wedge \ \mathit{IsSorted}[a^* \frown (U^* \frown b^*)] \tag{11}$$

and the current solution for V^* is $a^* \frown (U^* \frown b^*)$. By inference rule **IR-3** (reduce composite argument) using **Definition 1** the goal (11) becomes:

$$\begin{aligned}\mathcal{M}[a^* \frown (U^* \frown b^*)] = \mathcal{M}[X_0] \ \wedge \ U^* \leq a^* \ \wedge \\ \mathit{IsSorted}[U^*] \ \wedge \ b^* \leq U^*.\end{aligned} \tag{12}$$

By the equality in (12), U^* is smaller in the Noetherian ordering than X_0, thus by **ST-2** (induction) U^* replaced by $\mathit{MMSort}[W^*]$, the goal becomes:

$$\begin{aligned}\mathcal{M}[a^* \frown (\mathit{MMSort}[W^*] \frown b^*)] = \mathcal{M}[X_0] \ \wedge \ \mathit{Sort}[W^*] \leq a^* \\ \wedge \ \mathit{IsSorted}[\mathit{MMSort}[W^*]] \ \wedge \ b^* \leq \mathit{MMSort}[W^*]\end{aligned} \tag{13}$$

the intermediate solution for V^* is $a^* \frown (\mathit{Sort}[W^*] \frown b^*)$, and the corresponding induction hypothesis (with $X_0 \rightarrow W^*$ and $V \rightarrow \mathit{MMSort}[W^*]$) is added to the assumptions:

$$\mathcal{M}[\mathit{MMSort}[W^*]] = \mathcal{M}[W^*] \ \wedge \ \mathit{IsSorted}[\mathit{MMSort}[W^*]]. \tag{14}$$

The second conjunct of this assumption is used to reduce the goal (13) by rule **IR-2** to:

$$\begin{aligned}\mathcal{M}[a^* \frown (\mathit{MMSort}[W^*] \frown b^*)] = \mathcal{M}[X_0] \\ \wedge \ a^* \leq \mathit{Sort}[W^*] \ \wedge \ \mathit{MMSort}[W^*] \leq b^*.\end{aligned} \tag{15}$$

The first conjunct of (14) is used by **IR-7** (use equivalence) to reduce the previous goal to:

$$\mathcal{M}[a^* \frown (W^* \frown b^*)] = \mathcal{M}[X_0] \ \wedge \ a^* \leq W^* \ \wedge \ W^* \leq b^*. \tag{16}$$

By inference rule **IR-5** this becomes:

$$\{\!\{a^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{b^*\}\!\} = \mathcal{M}[X_0] \ \wedge \ a^* \leq W^* \ \wedge \ W^* \leq b^*. \tag{17}$$

The strategy **ST-3** (cascading) is applied to this goal and generates the conjecture:

Conjecture 4. $\underset{X}{\forall}\Big((X \neq \langle\rangle \ \wedge \ \mathit{Tail}[X] \neq \langle\rangle) \Longrightarrow \underset{a\,b\,U}{\exists\exists\exists}(\{\!\{a\}\!\} \uplus \mathcal{M}[W] \uplus \{\!\{b\}\!\} = \mathcal{M}[X] \ \wedge \ a \leq U \ \wedge \ U \leq b)\Big)$.

Proof 2 synthesizes the auxiliary functions $\mathit{min}[X], \mathit{max}[X]$ and $\mathit{Trim}[X]$ which split a list into its minimum, maximum and the rest, thus they have the property

Property 4. $\underset{X}{\forall}\Big((X \neq \langle\rangle \ \wedge \ \mathit{Tail}[X] \neq \langle\rangle) \Longrightarrow (\{\!\{\min[X]\}\!\} \uplus \mathcal{M}[\mathit{Trim}[X]] \uplus \{\!\{\max[X]\}\!\} = \mathcal{M}[X] \ \wedge \ \min[X] \leq \mathit{Trim}[X] \ \wedge \ \mathit{Trim}[X] \leq \max[X])\Big)$.

By strategy **ST-3** and inference rule **IR-4** the solutions of (17) are $a^* \to \mathit{min}[X_0]$, $b^* \to \mathit{max}[X_0]$, and $W^* \to \mathit{Trim}[X_0]$, which are substituted in the solution for V^* to give $\mathit{min}[X_0] \backsim (\mathit{Trim}[X_0] \frown \mathit{max}[X_0])$. *QED*

With this **Proof 1** is finished and the synthesized algorithm is[2]:

Algorithm 1. Min-Max-Sort.

$$\begin{pmatrix} \mathit{MMSort}[\langle\rangle] = \langle\rangle \\ \mathcal{M}[a \backsim \langle\rangle] = a \backsim \langle\rangle \\ (U \neq \langle\rangle \ \wedge \ \mathit{Tail}[U] \neq \langle\rangle) \Longrightarrow \\ \mathit{MMSort}[U] = \mathit{min}[U] \backsim (\mathit{MMSort}[\mathit{Trim}[U]] \frown \mathit{max}[U]) \end{pmatrix}$$

B. Synthesis of the Auxiliary Functions

The following proof of **Conjecture 4** synthesizes the auxiliary functions *min*, *max*, and *Trim* which are necessary in the main sorting function. This proof triggers the synthesis of further auxiliary functions *minA*, *maxA*, and *TrimA*.

Proof 2:

By natural style inference rules, take X_0 arbitrary but fixed, assume:

$$X_0 \neq \langle\rangle \tag{18}$$

$$\mathit{Tail}[X_0] \neq \langle\rangle \tag{19}$$

and prove:

$$\underset{x\,y\,W}{\exists\exists\exists}(\{\!\{x\}\!\} \uplus \mathcal{M}[W] \uplus \{\!\{y\}\!\} = \mathcal{M}[X_0] \ \wedge \ x \leq W \ \wedge \ W \leq y. \tag{20}$$

By introduction of metavariables the goal becomes:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^*. \tag{21}$$

Apply **ST-1** (cover set) to X_0, using only $a_0 \backsim (V_0 \frown b_0)$ because (18) and (19). The goal becomes:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \mathcal{M}[a_0 \backsim (V_0 \frown b_0)] \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^*. \tag{22}$$

[2]In the presentation of the algorithms it is assumed that all variables are universally quantified over their respective domains, according to our notation convention.

By rule **IR-8** we consider two cases depending on the ordering between a_0 and b_0:

Case 1.1. $a_0 \leq b_0$:

Apply **ST-3** (cascading) to generate the conjecture:

Conjecture 5.

$$\underset{X\,a\,b}{\forall\forall\forall}(a \leq b \Longrightarrow (\underset{x\,y\,W}{\exists\exists\exists}(\{\!\{x\}\!\} \uplus \mathcal{M}[W] \uplus \{\!\{y\}\!\} = \mathcal{M}[a \backsim (X \frown b)] \ \wedge \ x \leq W \ \wedge \ W \leq y))$$

Proof 3 synthesizes *minA*, *maxA* and *TrimA* which have the property:

Property 5.

$$\underset{a\,b\,X}{\forall\forall\forall}(a \leq b \Longrightarrow (\{\!\{\mathit{minA}[a,X,b]\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a,X,b]] \uplus \{\!\{\mathit{maxA}[a,X,b]\}\!\} = \mathcal{M}[a \backsim (X \frown b)] \ \wedge \ \mathit{minA}[a,X,b] \leq \mathit{TrimA}[a,X,b] \ \wedge \ \mathit{TrimA}[a,X,b] \leq \mathit{maxA}[a,X,b]))$$

By strategy **ST-3** (cascading) the goal (22) becomes:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \{\!\{\mathit{minA}[a,X,b]\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a,X,b]] \uplus \{\!\{\mathit{maxA}[a,X,b]\}\!\} \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^*. \tag{23}$$

and the following assumption is generated:

$$\mathit{minA}[a_0,X_0,b_0] \leq \mathit{TrimA}[a_0,X_0,b_0] \ \wedge \ \mathit{TrimA}[a_0,X_0,b_0] \leq \mathit{maxA}[a_0,X_0,b_0] \tag{24}$$

By strategy **ST-6** and inference rule **IR-4** the proof succeeds on this branch, and the solutions are $a^* \to \mathit{minA}[a_0,V_0,b_0]$, $b^* \to \mathit{maxA}[a_0,V_0,b_0]$, and $W^* \to \mathit{TrimA}[a_0,V_0,b_0]$.

Case 1.2. $b_0 < a_0$:

Property 5 is reused to modify the goal and similarly the solutions are $a^* \to \mathit{minA}[b_0,V_0,a_0]$, $b^* \to \mathit{maxA}[b_0,V_0,a_0]$, and $W^* \to \mathit{TrimA}[b_0,V_0,a_0]$.

QED

The extracted algorithm is:

Algorithm 2. Maximum, minimum, trim maximum and mimimum.

$$\begin{pmatrix} \mathit{min}[a \backsim (U \frown b)] = \begin{cases} \mathit{minA}[a,U,b], & \mathit{if}\ a \leq b \\ \mathit{minA}[b,U,a], & \mathit{if}\ b < a \end{cases} \\ \mathit{max}[a \backsim (U \frown b)] = \begin{cases} \mathit{maxA}[a,U,b], & \mathit{if}\ a \leq b \\ \mathit{maxA}[b,U,a], & \mathit{if}\ b < a \end{cases} \\ \mathit{Trim}[a \backsim (U \frown b)] = \begin{cases} \mathit{TrimA}[a,U,b], & \mathit{if}\ a \leq b \\ \mathit{TrimA}[b,U,a], & \mathit{if}\ b < a \end{cases} \end{pmatrix}$$

The following proof of **Conjecture 5** synthesizes the auxiliary functions used above.

Proof 3: By natural style rules and strategy **ST-2** (induction) the universal X is Skolemized to the *target constant* X_0, producing the *target goal*:

$$\underset{a\,b}{\forall\forall}\ a \leq b \implies \underset{x\,y\,W}{\exists\exists\exists}(\{\!\{x\}\!\} \uplus \mathcal{M}[W] \uplus \{\!\{y\}\!\} = \mathcal{M}[a \smile (X_0 \frown b)] \ \wedge \ x \leq W \ \wedge \ W \leq y). \tag{25}$$

Furthermore the rest of the universal variables become Skolem constants and the LHS of the implication is assumed:

$$a_0 \leq b_0 \tag{26}$$

The existential variables become metavariables and the current goal is:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \mathcal{M}[a_0 \smile (X_0 \frown b_0)] \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^* \tag{27}$$

which is extended using transitivity of $\leq$ and multiset decomposition to:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \{\!\{a_0\}\!\} \uplus \mathcal{M}[X_0] \uplus \{\!\{b_0\}\!\} \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^* \ \wedge \ x^* \leq y^*. \tag{28}$$

Strategy **ST-1** (cover set) is applied to X_0 with the cover set $\{\langle\rangle, c_0 \smile U_0\}$.
Case 1. $X_0 = \langle\rangle$:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \{\!\{a_0\}\!\} \uplus \mathcal{M}[\langle\rangle] \uplus \{\!\{b_0\}\!\} \wedge x^* \leq W^* \ \wedge \ W^* \leq y^* \ \wedge \ x^* \leq y^*. \tag{29}$$

Using equation split the metavariables are solved to:
$x^* \to a_0, y^* \to b_0, W^* \to \langle\rangle$.
Case 2. $X_0 = c_0 \smile U_0$:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \{\!\{a_0\}\!\} \uplus \mathcal{M}[c_0 \smile U_0] \uplus \{\!\{b_0\}\!\} \wedge x^* \leq W^* \ \wedge \ W^* \leq y^* \ \wedge \ x^* \leq y^*. \tag{30}$$

By multiset expansion the goal is transformed into:

$$\{\!\{x^*\}\!\} \uplus \mathcal{M}[W^*] \uplus \{\!\{y^*\}\!\} = \{\!\{a_0\}\!\} \uplus \{\!\{c_0\}\!\} \uplus \mathcal{M}[U_0] \uplus \{\!\{b_0\}\!\} \wedge x^* \leq W^* \ \wedge \ W^* \leq y^* \ \wedge \ x^* \leq y^*. \tag{31}$$

Since U_0 is smaller than X_0 w.r.t. the Noetherian ordering, by **ST-2** (induction) we assume the new induction hypothesis obtained from (25) by substituting $X_0 \to U_0$ and using the names of the desired functions:

$$\underset{a\,b}{\forall\forall}\ a \leq b \implies (\{\!\{\mathit{minA}[a, U_0, b]\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a, U_0, b]] \uplus \{\!\{\mathit{maxA}[a, U_0, b]\}\!\} = \{\!\{a\}\!\} \uplus \mathcal{M}[U_0] \uplus \{\!\{b\}\!\} \ \wedge \ \mathit{minA}[a, U_0, b] \leq \mathit{TrimA}[a, U_0, b] \wedge \mathit{TrimA}[a, U_0, b] \leq \mathit{maxA}[a, U_0, b] \wedge \mathit{minA}[a, U_0, b] \leq \mathit{maxA}[a, U_0, b]). \tag{32}$$

This is further instantiated on a, b as shown below.

By **IR-8** because (26) we consider three cases for c_0:
Case 2.1. $c_0 < a_0$: We instantiate (32) with $a \to c_0, b \to b_0$ and the proof proceeds similarly to the next case, finding the solutions: $x^* \to \mathit{minA}[c_0, U_0, b_0]$, $y^* \to \mathit{maxA}[c_0, U_0, b_0]$, $W^* \to a_0 \smile \mathit{TrimA}[c_0, U_0, b_0]$.
Case 2.2. $a_0 \leq c_0 \leq b_0$: (32) is instantiated with $a \to a_0, b \to b_0$, the resulting conjunction is split and forward inference is applied using (26). The resulting multiset equality is used in the goal (31) to obtain the new goal:

$$\{\!\{\mathit{minA}[a_0, U_0, b_0]\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a_0, U_0, b_0]] \uplus \{\!\{\mathit{maxA}[a_0, U_0, b_0]\}\!\} = \{\!\{\mathit{minA}[a_0, U_0, b_0]\}\!\} \uplus \{\!\{c_0\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a_0, U_0, b_0]] \uplus \{\!\{\mathit{maxA}[a_0, U_0, b_0]\}\!\} \ \wedge \ x^* \leq W^* \ \wedge \ W^* \leq y^* \ \wedge \ x^* \leq y^*. \tag{33}$$

By splitting the equation, compressing the multiset term $\{\!\{c_0\}\!\} \uplus \mathcal{M}[\mathit{TrimA}[a_0, U_0, b_0]]$ and solving multiset equations the solutions are: $x^* \to \mathit{minA}[a_0, U_0, b_0]$, $y^* \to \mathit{maxA}[a_0, U_0, b_0]$, $W^* \to c_0 \smile \mathit{TrimA}[a_0, U_0, b_0]$ and the inequalities in the goal are removed by backward inference using the current assumptions.
Case 2.3. $b_0 < c_0$: (32) is instantiated with $a \to a_0, b \to b_0$ and the proof is similar, finding the solutions: $x^* \to \mathit{minA}[a_0, U_0, c_0]$, $y^* \to \mathit{maxA}[a_0, U_0, c_0]$, $W^* \to b_0 \smile \mathit{TrimA}[a_0, U_0, c_0]$. *QED*

The extracted algorithms from the proofs are:

Algorithm 3. *Auxiliary minimum.*

$$\begin{pmatrix} \mathit{minA}[a, \langle\rangle, b] = a \\ \mathit{minA}[a, c \smile U, b] = \\ \begin{cases} \mathit{minA}[c, U, b], & \text{if } c < a \\ \mathit{minA}[a, U, b], & \text{if } a \leq c \leq b \\ \mathit{minA}[a, U, c], & \text{if } b < c \end{cases} \end{pmatrix}$$

Algorithm 4. *Auxiliary maximum.*

$$\begin{pmatrix} \mathit{maxA}[a, \langle\rangle, b] = b \\ \mathit{minA}[a, c \smile U, b] = \\ \begin{cases} \mathit{maxA}[c, U, b], & \text{if } c < a \\ \mathit{maxA}[a, U, b], & \text{if } a \leq c \leq b \\ \mathit{maxA}[a, U, c], & \text{if } b < c \end{cases} \end{pmatrix}$$

Note that both algorithms above can be simplified by retaining only two arguments and only two branches in the recursive step, however we keep this similar structure because we will combine them with the next algorithm.

Algorithm 5. *Auxiliary trim.*

$$\begin{pmatrix} \mathit{TrimA}[a, \langle\rangle, b] = \langle\rangle \\ \mathit{TrimA}[a, c \smile U, b] = \\ \begin{cases} a \smile \mathit{TrimA}[c, U, b], & \text{if } c < a \\ c \smile \mathit{TrimA}[a, U, b], & \text{if } a \leq c \ \wedge \ c \leq b \\ b \smile \mathit{TrimA}[a, U, c], & \text{if } b < c \end{cases} \end{pmatrix}$$

The function above has the same property as the bubble sort algorithm: while scanning the list it moves the small elements towards the beginning of the list and the big elements towards the end of the list, thus it contributes to making the list "more sorted". This suggests to transform this sorting algorithm into an imperative one, which would be basically the same as bubble sort algorithm, but working in both directions.

IV. Synthesis of Bidirectional Bubble Sort

A. The Tail Recursive Algorithm

First we transform auxiliary trimming algorithm *TrimA* into a tail–recursive one (the other two are already), namely into *TrimATR*. by using an additional argument as accumulator of the result. Since the input is scanned using $\backsim$, if we want to keep the same order of the elements than the result must be accumulated using $\frown$. (Since the only restriction on the result is related to its ordering w. r. t. some elements, the order of the elements in the result is not important, thus the more efficient $\backsim$ can also be used for accumulation, however we may want to keep the order of the element in order to preserve a nice practical property of the bubble sort algorithm: when the input is almost sorted, than the main loop has fewer executions.)

Algorithm 6. *Tail–recursive trim.*

$$\left(\begin{array}{c} \mathit{TrimATR}[a, \langle\rangle, b, V] = V \\ \mathit{TrimATR}[a, c \backsim U, b, V] = \\ \quad \begin{cases} \mathit{TrimATR}[c, U, b, V \frown a], & \text{if } c < a \\ \mathit{TrimATR}[a, U, b, V \frown c], & \text{if } a \leq c \ \wedge \ c \leq b \\ \mathit{TrimATR}[a, U, c, V \frown b], & \text{if } b < c \end{cases} \end{array}\right)$$

Because their structure is the same, the three tail–recursive algorithms obtained above can be combined into one, which returns a triple of elements (minimum, trimed list, and maximum), let us call this *mTm*, which uses an auxiliary tail–recursive *mTmA*.

Algorithm 7. *Min–trim–max in one function*

$$\left(\begin{array}{c} \mathit{mTm}[a \backsim (U \frown b)] = \begin{cases} \mathit{mTmA}[a, U, b, \langle\rangle], & \text{if } a \leq b \\ \mathit{mTmA}[b, U, a, \langle\rangle], & \text{if } b < a \end{cases} \\ \mathit{mTmA}[a, \langle\rangle, b, V] = \langle a, V, b\rangle \\ \mathit{mTmA}[a, c \backsim U, b, V] = \\ \begin{cases} \mathit{mTmA}[c, U, b, V \frown a], & \text{if } c < a \\ \mathit{mTmA}[a, U, b, V \frown c], & \text{if } a \leq c \ \wedge \ c \leq b \\ \mathit{mTmA}[a, U, c, V \frown b], & \text{if } b < c \end{cases} \end{array}\right)$$

Using this we can express the sorting algorithm in the following way:

Algorithm 8. Min-Max-Sort using tail recursive auxiliary function.

$$\left(\begin{array}{l} \mathit{MMSort}[\langle\rangle] = \langle\rangle \\ \mathit{MMSort}[a \backsim \langle\rangle] = a \backsim \langle\rangle \\ \mathit{MMSort}[a \backsim U \frown b] = \\ \qquad \mathit{MMSortA}[\mathit{mTm}[a \backsim U \frown b]] \\ \mathit{MMSortA}[\langle a, \langle\rangle, b\rangle] = a \backsim \langle\rangle \frown b \\ \mathit{MMSortA}[\langle a, c \backsim \langle\rangle, b\rangle] = a \backsim c \backsim \langle\rangle \frown b \\ \mathit{MMSortA}[\langle a, U, b\rangle] = \\ \qquad a \backsim \mathit{MMSortA}[\mathit{mTm}[U]] \frown b)] \end{array}\right)$$

This can also be transformed into a tail recursion using two accumulators and the function *Conc* for concatenation of lists:

Algorithm 9. Min-Max-Sort tail recursive.

$$\left(\begin{array}{l} \mathit{MMSort}[\langle\rangle] = \langle\rangle \\ \mathit{MMSort}[a \backsim \langle\rangle] = a \backsim \langle\rangle \\ \mathit{MMSort}[a \backsim U \frown b] = \\ \qquad \mathit{MMSortA}[\langle\rangle, \mathit{mTm}[a \backsim U \frown b], \langle\rangle] \\ \mathit{MMSortA}[V, \langle a, \langle\rangle, b\rangle, W] = \mathit{Conc}[V, a \backsim b \backsim W] \\ \mathit{MMSortA}[V, \langle a, c \backsim \langle\rangle, b\rangle, W] = \\ \qquad \mathit{Conc}[V, a \backsim c \backsim b \backsim W] \\ \mathit{MMSortA}[V, \langle a, U, b\rangle, W] = \\ \qquad \mathit{MMSortA}[V \frown a, \mathit{mTm}[U], b \backsim W] \end{array}\right)$$

B. Adding a flag for efficiency

The call from the last clause of **Algorithm 9** is not necessary in case $\mathit{mTm}[U]$ is already sorted, one can detect by logical analysis that this happens if in **Algorithm 7** the last clause follows the condition $a \leq c \ \wedge \ c \leq b$ during all recursive calls of *mTmA*. This suggests to add a flag to the arguments of *mTmA* in order to check this condition. The new algorithm is:

Algorithm 10. *Min–trim–max with flag*

$$\left(\begin{array}{c} \mathit{mTm}[a \backsim (U \frown b)] = \begin{cases} \mathit{mTmA}[a, U, b, \langle\rangle, \mathbb{T}], & \text{if } a \leq b \\ \mathit{mTmA}[b, U, a, \langle\rangle, \mathbb{T}], & \text{if } b < a \end{cases} \\ \mathit{mTmA}[a, \langle\rangle, b, V, f] = \langle a, V, b, f\rangle \\ \mathit{mTmA}[a, c \backsim U, b, V, f] = \\ \begin{cases} \mathit{mTmA}[c, U, b, V \frown a, \mathbb{F}], & \text{if } c < a \\ \mathit{mTmA}[a, U, b, V \frown c, f], & \text{if } a \leq c \ \wedge \ c \leq b \\ \mathit{mTmA}[a, U, c, V \frown b, \mathbb{F}], & \text{if } b < c \end{cases} \end{array}\right)$$

The corresponding sorting algorithm with flag is:

Algorithm 11. Min-Max-Sort with flag.

$$\left(\begin{array}{l}
\mathit{MMSort}[\langle\rangle] = \langle\rangle \\
\mathit{MMSort}[a \backsim \langle\rangle] = a \backsim \langle\rangle \\
\mathit{MMSort}[a \backsim U \frown b] = \\
\qquad \mathit{MMSortA}[\langle\rangle, \mathit{mTm}[a \backsim U \frown b], \langle\rangle] \\
\mathit{MMSortA}[V, \langle a, \langle\rangle, b, f\rangle, W] = \mathit{Conc}[V, a \backsim b \backsim W] \\
\mathit{MMSortA}[V, \langle a, c \backsim \langle\rangle, b, f\rangle, W] = \\
\qquad \mathit{Conc}[V, a \backsim c \backsim b \backsim W] \\
\mathit{MMSortA}[V, \langle a, U, b, \mathbb{T}\rangle, W] = \\
\qquad \mathit{Conc}[V, a \backsim U, b \backsim W] \\
\mathit{MMSortA}[V, \langle a, U, b, \mathbb{F}\rangle, W] = \\
\qquad \mathit{MMSortA}[V \frown a, \mathit{mTm}[U], b \backsim W]
\end{array}\right)$$

V. Conclusion

This paper demonstrates the possibility of synthesizing on a logical basis of the *Min-Max-Sort* algorithm, which is essentially a version of bubble sort with bidirectional accumulation of extreme values (smaller left-hand-side and greater right-hand-side). This kind of synthesis appears to be challenging because bubble sort looks much like a purely imperative algorithm – scanning the list and swapping some elements in certain cases. However, as we can see in our approach, by applying the cascading principle in the appropriate way and by adding tail recursion using accumulators, one can infer logically first a recursive algorithm, and then transform it into a tail recursive one.

Derivation of the tail recursive algorithm is interesting because, as also demonstrated in [10], such an algorithm can easily be transformed into a functional algorithm and then into an imperative algorithm, whose execution is in general much more efficient.

References

[1] D. R. Barstow. Remarks on "a synthesis of several sorting algorithms" by John Darlington. *Acta Informatica*, 13:225–227, 1980.

[2] B. Buchberger, T. Jebelean, T. Kutsia, A. Maletzky, and W. Windsteiger. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. *Journal of Formalized Reasoning*, 9(1):149–185, 2016.

[3] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: meta-level guidance for mathematical reasoning*. Cambridge University Press, 2005.

[4] R. M. Burstall and John Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, 1977.

[5] J. Darlington. *A semantic approach to automatic program improvement*. PhD thesis, University of Edinburg, 1972.

[6] J. Darlington. A synthesis of several sorting algorithms. *Acta Informatica*, 11:1–30, 1978.

[7] J. Darlington and R. M. Burstall. A System Which Automatically Improves Programs. *Acta Informatica*, 6(1):41–60, 1976.

[8] I. Dramnesc and T. Jebelean. Synthesis of List Algorithms by Mechanical Proving. *Journal of Symbolic Computation*, 68:61–92, 2015.

[9] I. Dramnesc and T. Jebelean. Proof–Based Synthesis of Sorting Algorithms Using Multisets in *Theorema*. In *FROM 2019*, pages 76–91. EPTCS 303, 2019.

[10] I. Dramnesc and T. Jebelean. Deductive Synthesis of Bubble-Sort Using Multisets. In *SAMI 2020*, 2020. In print.

[11] I. Dramnesc, T. Jebelean, and S. Stratulat. Mechanical Synthesis of Sorting Algorithms for Binary Trees by Logic and Combinatorial Techniques. *Journal of Symbolic Computation*, 90:3–41, 2019.

[12] R. Geoff Dromey. Derivation of sorting algorithms from a specification. *Computer Journal*, 30(6):512–518, 1987.

[13] Brian T. Howard. Another iteration on "A synthesis of several sorting algorithms", 1994.

[14] Yomna Ben Jmaa, Rabie Ben Atitallah, David Duvivier, and Maher Ben Jemaa. A Comparative Study of Sorting Algorithms with FPGA Acceleration by High Level Synthesis. *Computación y Sistemas*, 23:213, 2019.

[15] Yulia Korukhova. An Approach to Automatic Deductive Synthesis of Functional Programs. *Annals of Mathematics and Artificial Intelligence*, 50(3-4):255–271, 2007.

[16] K. K. Lau. Top-down synthesis of sorting algorithms. *The Computer Journal*, 35:A001–A007, 1992.

[17] Yanhong A. Liu and Scott D. Stoller. From Recursion to Iteration: What Are the Optimizations? *SIGPLAN Not.*, 34(11):73–82, 1999.

[18] Z. Manna and R. Waldinger. A Deductive Approach to Program Synthesis. *ACM Transactions on Programming Languages and System*, 2(1):90–121, 1980.

[19] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 1: Deductive Reasoning. Addison-Wesley, 1985.

[20] Z. Manna and R. Waldinger. Fundamentals Of Deductive Program Synthesis. *IEEE Transactions on Software Engineering*, 18(8):674–704, 1992.

[21] S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. *SIGPLAN Not.*, 45(1):313–326, January 2010.

[22] Jonathan Traugott. Deductive Synthesis of Sorting Programs. *Journal of Symbolic Computation*, 7(6):533–572, 1989.

[23] M. F. Umar, E. U. Munir, S. A. Shad, and M. W. Nisar. Enhancement of Selection, Bubble and Insertion Sorting Algorithm. *Research Journal of Applied Sciences, Engineering and Technology*, 2:263–271, 2014.

[24] W. Windsteiger. Theorema 2.0: A System for Mathematical Theory Exploration. In *ICMS'2014*, volume 8592 of *LNCS*, pages 49–52, 2014.