Proof–Based Synthesis of Sorting Algorithms 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

Using multisets, we develop novel techniques for mechanizing the proofs of the synthesis conjectures for list–sorting algorithms, and we demonstrate them in the *Theorema* system. We use the classical principle of extracting the algorithm as a set of rewrite rules based on the witnesses found in the proof of the synthesis conjecture produced from the specification of the desired function (input and output conditions). The proofs are in natural style, using standard rules, but most importantly domain specific inference rules and strategies. In particular the use of multisets allows us to develop powerful strategies for the synthesis of arbitrarily structured recursive algorithms by general Noetherian induction, as well as for the automatic generation of the specifications of all necessary auxiliary functions (insert, merge, split), whose synthesis is performed using the same method.

1 Introduction

We present a comprehensive case study in the automated synthesis of list sorting algorithms: two main proofs produce the most popular sorting algorithms (min–sort, quick–sort, insert–sort, merge–sort) and trigger all the proofs necessary for producing the needed auxiliary functions for inserting, splitting, and merging. This is a continuation of our work on exploring in parallel the theories of multisets, lists, and binary trees, for the purpose of developing proof methods for the synthesis of algorithms on these domains. In one related paper [12] we already investigated algorithms for deletion from lists and binary trees using multisets.

We follow the proof–based approach to automated synthesis: 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, in form of conditional rewrite rules. The theoretical basis and the correctness of this scheme is well–known [6] and we used earlier in [11, 15].

For the experiments we use the *Theorema* system [5], in which the logical formulae and the inferences are presented in *natural style*[1], and which also allows to execute the synthesized algorithms.

Related work. The theory of *multisets* is well studied in the literature, including computational formalizations (see e. g. [17], 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 [18]. A systematic formalization of the theory of lists using multisets for the proofs of correctness of various sorting algorithms is mechanized in Isabelle/HOL[2], which however does not address the problem of algorithm synthesis. A valuable formalization in a previous version of *Theorema* [4], which includes the theory exploration and the synthesis of a sorting algorithm is presented

[1] That means a style similar to the one used by humans, but not *natural deduction*
[2] https://isabelle.in.tum.de/library/HOL/HOL-Library/Sorting_Algorithms.html

M. Marin, A. Crăciun (Eds.):
Working Formal Methods Symposium 2019 (FROM 2019)
EPTCS 303, 2019, pp. 76–91, doi:10.4204/EPTCS.303.6

in [3], which also constituted the starting point 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 general Noetherian induction and cover–set decomposition. In our previous work we study proof–based algorithm synthesis in the theories of lists [9], sets [10] and binary trees [13] separately [7], [8], [14], [11], [15].

Originality. In contrast to our early investigations, the current study uses multisets, which leads to a crucial improvement of the proof techniques. Also, the experiments are performed in the new version of the *Theorema* system [5, 19]. More importantly, we do not use here algorithm schemata or concrete induction principles, but only *general Noetherian induction* starting from a specific *cover set* (usually based on the inductive definition of lists). Namely, during the proof of a statement $P[t]$, for any t' (also ground term) which represents an object which is strictly smaller than the object represented by t in the Noetherian ordering, $P[t']$ can be added to the current assumptions. (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.) In our approach 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.

Moreover we develop a systematic approach to the *cascading* method pioneered in [2]: when the proof needs an auxiliary function which is not present in the knowledge, the prover constructs a conjecture synthesis statement which is used to obtain it. We have been using cascading manually for the case of lists in [11], and in this paper we present it as automatic proof technique and we illustrate it on several examples: all auxiliary algorithms are generated by cascading starting from the sorting synthesis proofs.

For the purposes above, **three novel inference rules** and **six novel strategies** are introduced.

2 Proof–Based Synthesis

2.1 Context

Notation. Square brackets are used 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 $\underset{X}{\exists}$.

Theory. We consider three types: *elements*, finite *lists*, and finite *multisets*.

Elements (denoted by a, b) of lists are any objects whose domain is totally ordered (notation $\leq$ and $<$). The ordering on elements is extended to orderings between an element and a list/multiset and between lists/multisets, by requiring that all elements of the composite object observe the ordering relation[3].

Multisets may contain the same elements several times. $\emptyset$ denotes the empty multiset, $\{\{a\}\}$ denotes the multiset having only the element a once. 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. $\mathscr{M}[U]$ denotes the multiset of elements of the list U.

Lists (denoted U, V, W) are either empty $\langle\rangle$ or of the form $a \frown U$, where $\frown$ is the operation of prepending an element to a list (like *cons* of *Lisp*). The multiset of a list observes:

Property 1. $$\underset{a,U}{\forall}\left(\begin{array}{c}\mathscr{M}[\langle\rangle] = \emptyset \\ \mathscr{M}[a \frown U] = \{\{a\}\} \uplus \mathscr{M}[U]\end{array}\right)$$

Sorted lists are defined by:

[3] Note that this introduces exceptions to antisymmetry and transitivity when the empty list/multiset is involved.

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

The *type of objects* is used by the prover, however for brevity we do not include the type inferencing details in the proofs. In this presentation we just use an implicit typing based on the notation convention.
Problem and Approach. The problem consists in finding the sorted version of a given list, however by our approach several sub–problems may appear and require auxiliary algorithms (merge, insert, split, etc.). The synthesized algorithm is extracted from the proof of the *synthesis conjecture* based on the function specification. For univariate functions the specification consists in an input condition $I[X]$ and an output condition $O[X,Y]$, and the conjecture is:

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

Likewise, for a bivariate function one has $I[X,Y]$, $O[X,Y,Z]$, and the conjecture:

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

2.2 Special Inference Rules and Strategies

Following natural style proving, we use *Skolem constants* (denoted with numerical underscore like V_1, a_0) introduced for universal goals, as well as *metavariables* (denoted with star power like V^*, b_1^*) introduced for existential goals. The prover uses classical inference rules (split ground conjunctions, rewrite by equality, etc.) as well as special rules appropriate for lists/multisets. Some of these rules are already experimented in our previous work, and from those we list here only the ones which are used explicitely in the proofs presented in the paper. The main contribution of this paper consists in the novel inference rules and strategies which construct the proofs necessary for the synthesis of sorting algorithms and their auxiliary functions, namely the inference rules: **IR-1**, **IR-2**, and **IR-8**, as well as the strategies: **ST-1**, **ST-2**, **ST-3**, **ST-4**, **ST-5**, and **ST-6**.

2.2.1 Inference Rules

IR-1: *Forward inference.* If a ground atomic assumption matches a part of another (typically universal) assumption, instantiate the later and replace in it the resulting copy of the ground assumption by the constant *True*, then simplify truth constants to produce a new assumption. It is used for instance in proving the goal (13) (after the instantiation with the witnesses) on the basis of assumption (14).
IR-2: *Backward inference.* Transform the goal using some assumption or a specific logical principle. If a ground atomic assumption matches a part of a ground or existential goal, instantiate the later and replace in it the resulting copy of the ground assumption by the constant *True*, then simplify truth constants to produce a new goal. A specific logical principle is used for backward inference on goals containing metavariables, namely the fact that a formula having the structure $\underset{x}{\exists} P[x]$ is a logical consequence of the formula $\underset{x}{\exists} P[f[x]]$. Example: transformation of (9) into (10).
IR-3: *Reduce composite argument.* This rule uses the current knowledge to transform parts of the goal or of the assumptions into atoms whose arguments contain no function symbols. Example: (8) and (9).
IR-4: *Solve metavariable.* When the goal is $\mathscr{M}[X^*] = \mathscr{M}[\mathscr{T}]$ for a ground term $\mathscr{T}$, infer $X^* = \mathscr{T}$. Example: formula (20). Sometimes this involves several intermediate steps – see (9) – (14).
IR-5: *Expand multiset.* In the goal, a multiset term with a composite argument is expanded by equality into several multiset terms. This is typically used when the argument contains cover–set constants,

because about these we do not have much information in the assumptions, but by treating them separately we can obtain objects having more properties, for instance by applying induction. Example: (15) – (16).
IR-6: *Compress multiset.* This is the dual of the previous rule, and it is tipically applied when the arguments contain function calls introduced by induction or by cascading. Example: (26) – (27).
IR-7: *Use equivalence.* Equality of the corresponding multisets induces an equivalence relation on lists, which is compatible with the ordering relations induced by the domain ordering, as well as with the function *Sort*. Therefore the prover can rewrite parts of the goal or of the assumptions by replacing equivalent lists or by inferring new relations on lists which are equivalent to lists already related. Example: (12) – (13).
IR-8: *Two constants.* If the current proof situation contains two Skolem constants representing domain elements, say a_0, b_0, then the prover generates two cases: $a_0 \leq b_0$ and $b_0 < a_0$. Example: after (39).

2.2.2 Strategies

ST-1: *Cover set.* This strategy organizes the structure of each synthesis conjecture proof and the extraction of the synthesized algorithm. Each conjecture for the synthesis of a *target function* is a quantified statement over some *main universal variable*. A *cover set* is a set of universal terms[4] which represent the domain of the main universal variable, as described in [15].

We project this concept on Skolem constants: first the main universal variable is Skolemized ("arbitrary but fixed") — we call this the *target constant*, and we call the corresponding Skolemized goal the *target goal* – and then the corresponding cover–set terms are also grounded by Skolemization, we call these the *cover-set terms* and the corresponding constants the *cover-set constants*. The proof starts with a certain cover set (typically the one suggested by the recursive definition of the domain), and starts a proof branch for each ground term ("proof by cases") – see *Alternative 2* in **Proof 1**. On each proof branch the input conditions of the function are assumed, and then the existential variable corresponding to the output value of the function is transformed into a metavariable whose value (the "witness") will be found on the respective branch of the proof. Finally the algorithm will be generated as a set of [conditional] equalities: the terms of the cover set become arguments ("patterns") on the LHS of the equalities, and the corresponding witnesses become the RHS of these, after replacing back the Skolem constants by variables. The strategy can be applied in a *nested* way, by choosing a new target constant among the Skolem constants of the goal – see *Alternative 2.3* in **Proof 6**.

The strategy is applied similarly to a metavariable from the goal (see *Alternative 1* in **Proof 1**), here the variables of the cover–set terms are replaced by metavariables. If on some branch the cover–set term is constant (it contains no metavariables), then the solution is constant and it may impose certain conditions on the Skolem constants involved in the goal, which will be used as conditions on the inputs (which correspond to the respective Skolem constants) in the final expression of the algorithm. In order to ensure mutual exclusion, the negation of these conditions are transmitted as additional assumptions to the next branches – see formula (7).
ST-2: *Induction.* We use Noetherian induction based on the well–founded ordering between lists determined by the strict inclusion of the corresponding multisets. This ordering checked either syntactically by the meta-relation between terms induced by the strict inclusion of the multisets of constants occurring in the terms, either semantically by using the current assumptions: for instance if $a_0 \cup U_0$ is a cover–set term for the target constant X_0 then U_0 is smaller than X_0.

[4] Terms containing universally quantified variables, such that for every element of the domain there exists exactly one term in the set which instantiates to that element.

When a ground term t represents an object which is smaller than the target constant X_0 of the target goal $P[X_0]$, then $P[t]$ is added as a new assumption, but modified by inserting the corresponding call of the target function instead of the existential variable.

Example: the target function is $F[X,Y]$, the target constant is X_0, the target goal $P[X_0]$ is $\underset{Y}{\forall}\,(I[X_0,Y] \Longrightarrow \underset{Z}{\exists}\,O[X_0,Y,Z])$, and we have a ground term t smaller than X_0 in the well–founded ordering. The instance $P[t]$ of the target goal is $\underset{Y}{\forall}\,(I[t,Y] \Longrightarrow \underset{Z}{\exists}\,O[t,Y,Z])$, The prover adds the assumption $\underset{Y}{\forall}\,(I[t,Y] \Longrightarrow O[t,Y,F[t,Y]])$, Typically in the subsequent proof this will be instantiated with a ground term s, then $I[t,s]$ will be proven and $O[t,s,F[T,s]]$ will be obtained as assumption, leading to the replacement of some subterm[s] of the goal with $F[t,s]$. In this way the recursive calls of F are explicitly generated in the synthesized algorithm, see for instance formulae (57) to (58).

This strategy is applied in a similar manner to metavariables, when they occur in the goal. When a metavariable Y^* represents an object which is smaller than the target constant X_0, then $P[Y^*]$ may be added as new assumption – see formula (11).

ST-3: *Cascading.* This strategy consists in proving separately a conjecture for synthesizing the algorithm for some auxiliary functions needed in the current proof. The Skolem constants from the current goal become universal variables $x,x',\ldots$, the metavariables from the current goal become existential variables $y,y',\ldots$, and the conjecture has the structure[5]:

$$\underset{x}{\forall}\,\underset{x'}{\forall}\,\ldots(P[x,x',\ldots] \Longrightarrow \underset{y}{\exists}\,\underset{y'}{\exists}\,\ldots Q[x,x',\ldots,y,y',\ldots]) \tag{1}$$

$P[x,x',\ldots]$ is composed from the assumptions which contain *only* the Skolem constants present in the goal, and $Q[x,x',\ldots,y,y',\ldots]$ is composed from the goal. A successfull proof of the conjecture generates the functions $f[x,x',\ldots], f'[x,x',\ldots],\ldots$, which have the property:

$$\underset{x}{\forall}\,\underset{x'}{\forall}\,\ldots(P[x,x',\ldots] \Longrightarrow Q[x,x',\ldots,f[x,x',\ldots],f'[x,x',\ldots],\ldots]) \tag{2}$$

The current proof continues after adding this property to the assumptions, thus if some of the generated functions are necessary later in the proof, they can be used without a new cascading step. Similar to the situation described at **ST-2**, the new assumption will trigger the simplification of the current goal by inserting the auxiliary function – see for instance formulae (19) and (20).

ST-4: *Pair multisets.* This strategy applies when the goal contains an equality of the shape: $\mathscr{M}[Y^*] = \mathscr{M}[t_1] \uplus \mathscr{M}[t_2] \uplus \ldots$, where Y^* is the metavariable we need to solve, and $t_1,t_2,\ldots$ are ground terms. A typical flow of the proof consists in transforming the union on the RHS of the equality into a single $\mathscr{M}[t]$, because this gives the solution $Y^* \to t$. To this effect the prover groups pairs of operands of $\uplus$ together (no matter whether they are contingent or not, because commutativity), creating alternatives for different groupings. For each pair a conjecture is created as described at strategy **ST-3** (cascading), from which a multiset term which equals the union of the pair can be constructed in one of the following ways:

– the auxiliary function is already known, the proof works by predicate logic;

– induction can be applied (if the target function is binary) - see formula (60);

– a separate synthesis proof of the function is necessary by **ST-3** (cascading) – see **Conjecture 5**.

ST-5: *Split.* When a union of multisets in the RHS of the goal must be sorted and it contains $\{\{a\}\}$ and $\mathscr{M}[X]$ where a and X are incomparable, split X into X_1,X_2 such that $X_1 \leq a$ and $a < X_2$. Similarly

[5]By local convention, here x,x',y,y' represent any kind of objects: domain elements or lists.

to the situation shown at **ST-4**, the two lists are found either by already known auxiliary functions, by induction, or by cascading, and the goal is updated appropriately with the corresponding terms. Example: *Alternative 2.2.2* in **Proof 1**.

ST-6: *Split goal equation.* When the goal contains several metavariables in an equation, then split the equation into several ones, such that only one metavariable occurs in every new equation. Uses heuristics to match the appropriate values. Example: formulae (42) and (43).

3 Synthesis of Sorting

The experiments start with the synthesis of *sorting* — the target function is *Sort*. By cascading this will trigger the synthesis of other auxiliary algorithms for insertion, merging, and splitting. According to **Conjecture 1** the synthesis conjecture is:

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

Proof 1: *Sort list by definition–based cover set.*
Universal X is Skolemized to target constant X_0, producing the target goal:

$$\underset{V}{\exists}\,\mathscr{M}[V] = \mathscr{M}[X_0] \ \wedge\ \mathit{IsSorted}[V] \tag{3}$$

and the existential V becomes the metavariable V^*:

$$\mathscr{M}[V^*] = \mathscr{M}[X_0] \ \wedge\ \mathit{IsSorted}[V^*]. \tag{4}$$

Two alternatives are pursued, by applying strategy **ST-1** (cover set) to the metavariable V^* or to the Skolem constant X_0 :
Alternative 1: Apply **ST-1** (cover set) to V^* with the cover set determined by the domain definition: $\{\langle\rangle,\ a^* \frown U^*\}$

Case 1.1. $V^* = \langle\rangle$: The goal (4) becomes:

$$\mathscr{M}[\langle\rangle] = \mathscr{M}[X_0] \ \wedge\ \mathit{IsSorted}[\langle\rangle]. \tag{5}$$

By inference rule **IR-2** (backward inference) using **Definition 1** the goal (5) becomes:

$$\mathscr{M}[\langle\rangle] = \mathscr{M}[X_0]. \tag{6}$$

By **ST-1** (cover set) 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. $V^* = a^* \frown U^*$: The condition on X_0 from the previous branch is added as assumption:

$$X_0 \neq \langle\rangle. \tag{7}$$

The goal (4) becomes:

$$\mathscr{M}[a^* \frown U^*] = \mathscr{M}[X_0] \ \wedge\ \mathit{IsSorted}[a^* \frown U^*] \tag{8}$$

and the current solution for V^* is $a^* \frown U^*$. By inference rule **IR-3** (reduce composite argument) using **Definition 1** the goal (8) becomes:

$$\mathscr{M}[a^* \frown U^*] = \mathscr{M}[X_0] \ \wedge\ a^* \leq U^* \ \wedge\ \mathit{IsSorted}[U^*]. \tag{9}$$

By **IR-2** (backward inference) U^* is replaced by $Sort[W^*]$ and the goal becomes:

$$\mathscr{M}[a^* \frown Sort[W^*]] = \mathscr{M}[X_0] \ \wedge \ a^* \leq Sort[W^*] \ \wedge \ IsSorted[Sort[W^*]] \tag{10}$$

and the intermediate solution for V^* is $a^* \frown Sort[W^*]$. Since $a^* \frown Sort[W^*]$ stands for V^* which has the same elements as the target constant X_0, the prover infers that W^* is less than X_0 in the well–founded ordering, thus by strategy **ST-2** (induction) the target goal (3) is used with $\{X \to W^*\}$ and $\{V \to Sort[W^*]\}$ to generate the assumption:

$$\mathscr{M}[Sort[W^*]] = \mathscr{M}[W^*] \ \wedge \ IsSorted[Sort[W^*]]. \tag{11}$$

The second conjunct of this assumption is used to reduce the goal (10) by rule **IR-2** to:

$$\mathscr{M}[a^* \frown Sort[W^*]] = \mathscr{M}[X_0] \ \wedge \ a^* \leq Sort[W^*]. \tag{12}$$

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

$$\mathscr{M}[a^* \frown W^*] = \mathscr{M}[X_0] \ \wedge \ a^* \leq W^*. \tag{13}$$

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

Conjecture 4. $\underset{X}{\forall}(X \neq \langle\rangle \Longrightarrow \underset{a}{\exists}\underset{U}{\exists}(\mathscr{M}[a \frown U] = \mathscr{M}[X] \ \wedge \ a \leq U))$.

Proof 3 synthesizes the functions $min[X]$ and $Trim[X]$ which split a list into its minimum and the rest. By **ST-3** (cascading) the new assumption is:

$$\underset{X}{\forall}(X \neq \langle\rangle \Longrightarrow (\mathscr{M}[min[X] \frown Trim[X]] = \mathscr{M}[X] \ \wedge \ min[X] \leq Trim[X])). \tag{14}$$

Using (7) this solves the goal (13) with the witnesses: $\{a^* \to min[X_0]\}$ and $\{W^* \to Trim[X_0]\}$, which gives for V^* the final solution $min[X_0] \frown Sort[Trim[X_0]]$. The algorithm extracted from the proof is:

Algorithm 1. Min-Sort.

$$\underset{U}{\forall}\left(\begin{array}{c} Sort[\langle\rangle] = \langle\rangle \\ U \neq \langle\rangle \Longrightarrow Sort[U] = min[U] \frown Sort[Trim[U]] \end{array}\right)$$

Alternative 2: Apply **ST-1** on X_0 with the cover set $\{\langle\rangle,\ a_0 \frown U_0\}$, starting two branches:

Case 2.1. $X_0 = \langle\rangle$ is straightforward. The solution is $\{V^* \to \langle\rangle\}$.

Case 2.2. $X_0 = a_0 \frown U_0$: The goal becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[a_0 \frown U_0] \wedge IsSorted[V^*]. \tag{15}$$

By **IR-5** (expand multiset) using **Property 1** the goal is transformed into:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \wedge IsSorted[V^*]. \tag{16}$$

Two alternatives are pursued, depending on the strategy used for this goal (**ST-2** or **ST-5**).

Alternative 2.2.1. Strategy **ST-2** (induction) uses U_0 (smaller than X_0) to produce the assumption:

$$\mathscr{M}[Sort[U_0]] = \mathscr{M}[U_0] \ \wedge \ IsSorted[Sort[U_0]]. \tag{17}$$

The goal (16) is rewritten by equality (17) into:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[Sort[U_0]] \ \wedge \ IsSorted[V^*]. \tag{18}$$

Strategy **ST-4** (pair multisets) applied to $\{\{a\}\}$ and $\mathscr{M}[Sort[U_0]]$, using (17) and (18), produces the conjecture:

Conjecture 5. $\underset{a}{\forall}\underset{X}{\forall}(IsSorted[X] \implies \underset{V}{\exists}(\mathscr{M}[V] = \{\{a\}\} \uplus \mathscr{M}[X] \wedge IsSorted[V]))$.

The function $Insert[a,X]$ which inserts an element in a sorted list, keeping it sorted, is synthesized by the same method[6]. By strategy **ST-3** (cascading) the new assumption is:

$$\underset{a}{\forall}\underset{X}{\forall}(IsSorted[X] \implies (\mathscr{M}[Insert[a,X]] = \{\{a\}\} \uplus \mathscr{M}[X] \wedge IsSorted[Insert[a,X]])) \tag{19}$$

and the goal (18) becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[Insert[a_0, Sort[U_0]]] \wedge IsSorted[V^*]. \tag{20}$$

By **IR-4** (solve metavariable) the solution for V^* is $Insert[a_0, Sort[U_0]]$ and the proof succeeds by standard logical inferences, thus the algorithm is:

Algorithm 2. Insert-Sort.

$$\underset{a,U}{\forall}\begin{pmatrix} Sort[\langle\rangle] = \langle\rangle \\ Sort[a \frown U] = Insert[a, Sort[U]] \end{pmatrix}$$

Alternative 2.2.2. The RHS of the equality in the goal (16) represents a list which must be sorted and it contains $\{\{a_0\}\}$ and $\mathscr{M}[U_0]$, where a_0 and U_0 are incomparable by the current assumptions. Therefore the strategy **ST-5** (split) applies to generate the conjecture:

Conjecture 6. $\underset{a}{\forall}\underset{X}{\forall}\underset{V_1}{\exists}\underset{V_2}{\exists}(\mathscr{M}[X] = \mathscr{M}[V_1] \uplus \mathscr{M}[V_2] \ \wedge \ V_1 \leq a \ \wedge \ a < V_2)$.

Proof 5 of this conjecture generates the algorithms for the functions $SmEq[a,X]$ and $Bigger[a,X]$ which split the list X into two lists having elements which are smaller, respectively bigger than a.

By strategy **ST-3** (cascading) the new assumption is:

$$\underset{a}{\forall}\underset{X}{\forall}(\mathscr{M}[X] = \mathscr{M}[SmEq[a,X]] \uplus \mathscr{M}[Bigger[a,X]] \ \wedge \ SmEq[a,X] \leq a \ \wedge \ a < Bigger[a,X]). \tag{21}$$

By strategy **ST-5** (split) this is instantiated with a_0 and U_0 to produce:

$$\begin{aligned} \mathscr{M}[U_0] = \mathscr{M}[SmEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \ \wedge \\ SmEq[a_0,U_0] \leq a_0 \ \wedge \ a_0 < Bigger[a_0,U_0]). \end{aligned} \tag{22}$$

and the goal (16) is transformed into:

$$\mathscr{M}[V^*] = \mathscr{M}[SmEq[a_0,U_0]] \uplus \{\{a_0\}\} \uplus \mathscr{M}[Bigger[a_0,U_0]] \ \wedge \ IsSorted[V^*]. \tag{23}$$

[6]For space reasons the proof is not included in this paper.

Because (22) neither of $SmEq[a_0,U_0]$ and $Bigger[a_0,U_0]$ can have more elements than U_0 and this is smaller in the well–founded ordering than the target constant X_0 because it is a part of a cover–set term. Thus strategy **ST-2** (induction) is applied to both, producing assumptions:

$$\mathscr{M}[SmEq[a_0,U_0]] = \mathscr{M}[Sort[SmEq[a_0,U_0]]] \ \wedge \ IsSorted[Sort[SmEq[a_0,U_0]]], \tag{24}$$

$$\mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[Sort[Bigger[a_0,U_0]]] \ \wedge \ IsSorted[Sort[Bigger[a_0,U_0]]]. \tag{25}$$

Rewriting using (24) and (25) replaces in the goal (16) the corresponding subterms to obtain:

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[SmEq[a_0,U_0]]] \uplus \{\{a_0\}\} \uplus \mathscr{M}[Sort[Bigger[a_0,U_0]]] \wedge IsSorted[V^*]. \tag{26}$$

Using **IR-6** by **Property 1** this becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[SmEq[a_0,U_0]]] \uplus \mathscr{M}[a_0 \frown Sort[Bigger[a_0,U_0]]] \ \wedge \ IsSorted[V^*]. \tag{27}$$

By **IR-1** (forward inference) using the current assumptions and the properties of inequality the following are obtained: $Sort[SmEq[a_0,U_0]] \leq a_0 < Sort[Bigger[a_0,U_0]]]$, $IsSorted[a_0 \frown Sort[Bigger[a_0,U_0]]]$ and $Sort[SmEq[a_0,U_0]] \leq a_0 \frown Sort[Bigger[a_0,U_0]]$.
Strategy **ST-4** applied to $\mathscr{M}[Sort[SmEq[a_0,U_0]]]$ and $\mathscr{M}[a_0 \frown Sort[Bigger[a_0,U_0]]]$ produces:
Conjecture 7. $\underset{X}{\forall}\underset{Y}{\forall}((X \leq Y \wedge IsSorted[X] \wedge IsSorted[Y]) \Longrightarrow \underset{V}{\exists}(\mathscr{M}[V] = \mathscr{M}[X] \uplus \mathscr{M}[Y] \wedge IsSorted[V]))$.
The algorithm *Conc* which concatenates two lists into a sorted one, if the conditions are like above, is also synthesized by our prover[7]. The new goal is:

$$\mathscr{M}[V^*] = \mathscr{M}[Conc[Sort[SmEq[a_0,U_0]]], a_0 \frown Sort[Bigger[a_0,U_0]]] \ \wedge \ IsSorted[V^*], \tag{28}$$

which gives the obvious solution to V^* and the algorithm *Quick-Sort*:
Algorithm 3. Quick-Sort.

$$\underset{a,U}{\forall}\left(\begin{array}{c} Sort[\langle\rangle] = \langle\rangle \\ Sort[a \frown U] = Conc[Sort[SmEq[a,U]], a \frown Sort[Bigger[a,U]]] \end{array}\right)$$

QED

Another approach is to consider a cover set corresponding to the *divide–and–conquer* principle: $\{\langle\rangle,\ a \frown \langle\rangle,\ Conc[U,V]\}$ (where U,V are nonempty). Here *Conc* is used as a *pattern matching* construct, which may appear on the LHS of a rewrite rule, and it comes together with a simple splitting function, which gives two nonempty lists from a list having at least two elements. (For lack of space we omit here a possible splitting algorithm and its automatic generation by the principles presented in this paper.) The proof proceeds in a similar manner, with several alternatives and successful branches, from which we summarize below only the most interesting ones.
Proof 2: *Sort list by divide–and–conquer cover set.*
By quantified inferences the target goal is the same as in the previous proof:

$$\mathscr{M}[V^*] = \mathscr{M}[X_0] \ \wedge \ IsSorted[V^*]. \tag{29}$$

Alternative 1: Application of the cover–set strategy to metavariable V^* produces *Quick–Sort.*

[7]For lack of space the proof is not presented in this paper

Alternative 2: Application of the cover–set strategy to X_0. Cases $\langle\rangle$ and $a_0 \frown \langle\rangle$ are straightforward.
Case $X_0 = Conc[U_1, U_2]$: After splitting the multiset the goal becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[U_1] \uplus \mathscr{M}[U_2] \ \wedge \ IsSorted[V^*]. \tag{30}$$

After applying **ST-2** (induction)[8] on U_1 and on U_2 (we do not list the obvious assumptions):

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[U_1]] \uplus \mathscr{M}[Sort[U_2]] \ \wedge \ IsSorted[V^*]. \tag{31}$$

Strategy **ST-4** (pair multisets) produces the conjecture:

Conjecture 8.
$\underset{U_1,U_2}{\forall} (IsSorted[U_1] \wedge IsSorted[U_2] \Longrightarrow \underset{W}{\exists} (\mathscr{M}[W] = \mathscr{M}[U_1] \uplus \mathscr{M}[U_2] \ \wedge \ IsSorted[W])).$

The proofs in section 5 synthesize several algorithms for the function *Merge* which combines two sorted lists into a sorted one. The corresponding sorting algorithm is:

Algorithm 4. Merge Sort.

$$\underset{a,U,V}{\forall} \begin{pmatrix} Sort[\langle\rangle] = \langle\rangle \\ Sort[a \frown \langle\rangle] = a \frown \langle\rangle \\ Sort[Conc[U,V]] = Merge[Sort[U], Sort[V]] \end{pmatrix}$$

QED

4 Splitting

4.1 Split into minimum/rest of elements.

The target functions are $min[X]$ which selects from X the minimum element according to the domain ordering and $Trim[X]$ which gives the list without it. We need to prove **Conjecture 4**.
Proof 3: *Min and Trim.*
By natural style proving, take X_0 arbitrary but fixed, assume:

$$X_0 \neq \langle\rangle \tag{32}$$

and after introducing the existential metavariables, the goal is:

$$\mathscr{M}[X_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq X_0. \tag{33}$$

Strategy **ST-1** (cover set) applies to X_0, using only $a_0 \frown U_0$ because (32). The goal is:

$$\mathscr{M}[a_0 \frown U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \frown U_0. \tag{34}$$

By **IR-3** (composite argument) on the last conjunct the goal becomes:

$$\mathscr{M}[a_0 \frown U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \ \wedge \ y^* \leq U_0. \tag{35}$$

Strategy **ST-3** (cascading) generates the conjecture:

Conjecture 9. $\underset{X}{\forall} \underset{a}{\forall} \underset{y}{\exists} \underset{Y}{\exists} (\mathscr{M}[a \frown X] = \mathscr{M}[Y] \uplus \{\{y\}\} \ \wedge \ y \leq a \ \wedge \ y \leq X).$

[8] Note that induction can be applied only when U_1, U_2 are assumed nonempty.

Proof 4 synthesizes the auxiliary functions *minA* and *TrimA* which have the property:

$$\forall_X \forall_a (\mathscr{M}[a \frown X] = \mathscr{M}[TrimA[a,X]] \uplus \{\{minA[a,X]\}\} \ \wedge \ minA[a,X] \leq a \ \wedge \ minA[a,X] \leq X) \tag{36}$$

and which solves the goal (35) using the witnesses $\{Y^* \to TrimA[a_0,U_0],\ y^* \to minA[a_0,U_0]\}$. *QED*

We prove now **Conjecture 9**.

Proof 4: *Min and Trim auxiliary.*

By quantified inferences the goal becomes:

$$\mathscr{M}[a_0 \frown X_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \ \wedge \ y^* \leq X_0. \tag{37}$$

Apply **ST-1** (cover set) on X_0.

Case 1. $X_0 = \langle\rangle$ is straightforward, the solutions are: $\{y^* \to a_0,\ Y^* \to \langle\rangle\}$.

Case 2. $X_0 = b_0 \frown U_0$ generates the goal:

$$\mathscr{M}[a_0 \frown (b_0 \frown U_0)] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \ \wedge \ y^* \leq b_0 \frown U_0. \tag{38}$$

By **IR-5** (expand multiset) and **IR-3** (reduce composite argument) the goal becomes:

$$\{\{a_0\}\} \uplus \{\{b_0\}\} \uplus \mathscr{M}[U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \ \wedge \ y^* \leq b_0 \ \wedge \ y^* \leq U_0. \tag{39}$$

Two cases for domain element constants are generated by rule **IR-8** (two constants):

Case 2.1.

$$a_0 \leq b_0 \tag{40}$$

Strategy **ST-2** (induction) applies to U_0, a_0 in (37) and add the assumption:

$$\begin{aligned} \mathscr{M}[U_0] \uplus \{\{a_0\}\} = \mathscr{M}[TrimA[a_0,U_0]] \uplus \{\{minA[a_0,U_0]\}\} \quad \wedge \\ minA[a_0,U_0] \leq a_0 \quad \wedge \ minA[a_0,U_0] \leq U_0. \end{aligned} \tag{41}$$

(39) is rewritten by equality (41):

$$\begin{aligned} \mathscr{M}[TrimA[a_0,U_0]] \uplus \{\{minA[a_0,U_0]\}\} \uplus \{\{b_0\}\} = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \quad \wedge \\ y^* \leq a_0 \quad \wedge \quad y^* \leq b_0 \quad \wedge \quad y^* \leq U_0. \end{aligned} \tag{42}$$

The goal equation is split by strategy **ST-6**:

$$\begin{aligned} \mathscr{M}[TrimA[a_0,U_0]] \uplus \{\{b_0\}\} = \mathscr{M}[Y^*] \ \wedge \\ \{\{minA[a_0,U_0]\}\} = \{\{y^*\}\} \ \wedge \ y^* \leq a_0 \ \wedge \ y^* \leq b_0 \ \wedge \ y^* \leq U_0. \end{aligned} \tag{43}$$

By **IR-4** (solve metavariable) the solutions are: $\{y^* \to minA[a_0,U_0],\ Y^* \to b_0 \frown TrimA[a_0,U_0]\}$ and the remaining goal is proven by standard logic and properties of ordering.

Case 2.2.

$$b_0 < a_0 \tag{44}$$

The proof proceeds similarly by applying induction on U_0, b_0 in (37)) and the obtained solutions are: $\{y^* \to minA[b_0,U_0],\ Y^* \to a_0 \frown TrimA[b_0,U_0]\}$.

QED

The extracted algorithms from the proofs are:

Algorithm 5. Minimum.

$$\underset{a,b,U}{\forall}\left(\begin{array}{c} min[a \frown U] = minA[a,U] \\ minA[a,\langle\rangle] = a \\ minA[a,b \frown U] = \begin{cases} minA[a,U], & \text{if } a \le b \\ minA[b,U], & \text{if } b < a \end{cases} \end{array}\right)$$

Algorithm 6. Trim.

$$\underset{a,b,U}{\forall}\left(\begin{array}{c} Trim[a \frown U] = TrimA[a,U] \\ TrimA[a,\langle\rangle] = \langle\rangle \\ TrimA[a,b \frown U] = \begin{cases} b \frown TrimA[a,U], & \text{if } a \le b \\ a \frown TrimA[b,U], & \text{if } b < a \end{cases} \end{array}\right)$$

4.2 Split into smaller/bigger elements.

We need functions $SmEq[a,X]$ and $Bigger[a,X]$ which select from X the elements which are smaller or equal, respectively strictly bigger than a according to the domain ordering. We prove **Conjecture 6**.
Proof 5: *Split.*
a Skolemizes to a_0 and X to X_0 (target constant), and the goal uses the metavariables V^*, W^*:

$$\mathscr{M}[X_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \ \wedge\ V^* \le a_0 \ \wedge\ a_0 < W^*. \tag{45}$$

Strategy **ST-1** applies to X_0 with cover set $\{\langle\rangle,\ b_0 \frown U_0\}$:
Case 1. $X_0 = \langle\rangle$ is straightforward with solutions: $\{V^* \to \langle\rangle,\ W^* \to \langle\rangle\}$.
Case 2. $X_0 = b_0 \frown U_0$:

$$\mathscr{M}[b_0 \frown U_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \ \wedge\ V^* \le a_0 \ \wedge\ a_0 < W^*. \tag{46}$$

By **IR-5** (expand multiset):

$$\{\{b_0\}\} \uplus \mathscr{M}[U_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \ \wedge\ V^* \le a_0 \ \wedge\ a_0 < W^*. \tag{47}$$

By **ST-2** (induction) on U_0 (smaller than X_0) adds the assumption:

$$\mathscr{M}[U_0] = \mathscr{M}[SmEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \ \wedge\ SmEq[a_0,U_0] \le a_0 \ \wedge\ a_0 < Bigger[a_0,U_0]. \tag{48}$$

By rewriting $\mathscr{M}[U_0]$ in the goal:

$$\{\{b_0\}\} \uplus \mathscr{M}[SmEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \ \wedge\ V^* \le a_0 \ \wedge\ a_0 < W^*. \tag{49}$$

Inference rule **IR-8** (two constants) issues two cases:

Case 2.1.

$$b_0 \le a_0 \tag{50}$$

Strategy **ST-6** (split goal equation) changes the goal:

$$\{\{b_0\}\} \uplus \mathscr{M}[SmEq[a_0,U_0]] = \mathscr{M}[V^*] \ \wedge\ V^* \le a_0, \tag{51}$$

$$\mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[W^*] \ \wedge\ a_0 < W^*. \tag{52}$$

By **IR-6** (compress multiset) in (51), and by **IR-4** (solve metavariable) in both (51) and (52), the obtained solutions are: $\{V^* \to b_0 \frown SmEq[a_0,U_0],\ W^* \to Bigger[a_0,U_0]\}$ and the remaining goal is proven by standard inferences.

Case 2.2.

$$a_0 < b_0 \tag{53}$$

Similarly, the obtained solutions are: $\{V^* \to SmEq[a_0, U_0],\ W^* \to b_0 \smile Bigger[a_0, U_0]\}$.

QED

Algorithm 7. Small or equal

$$\underset{a,b,U}{\forall} \left(\begin{array}{c} SmEq[a, \langle\rangle] = \langle\rangle \\ SmEq[a, b \smile U] = \begin{cases} b \smile SmEq[a,U], & \text{if } b \leq a \\ SmEq[a,U], & \text{if } a < b \end{cases} \end{array} \right)$$

Algorithm 8. Bigger

$$\underset{a,b,U}{\forall} \left(\begin{array}{c} Bigger[a, \langle\rangle] = \langle\rangle \\ Bigger[a, b \smile U] = \begin{cases} Bigger[a,U], & \text{if } b \leq a \\ b \smile Bigger[a,U], & \text{if } a < b \end{cases} \end{array} \right)$$

5 Merging

For lack of space we cannot present here the synthesis proofs for *Insert* and *Conc*, the generated algorithms are the standard well known recursive ones. We focus instead on the merging of two sorted lists into a sorted one, which is more interesting because many alternative algorithms are produced.

Proof 6: *Merge.*

The goal **Conjecture 8** is Skolemized (X_0 is the target constant), and the target goal is:

$$\underset{Y}{\forall} \left((IsSorted[X_0] \wedge IsSorted[Y]) \Longrightarrow \underset{W}{\exists} \left(\mathscr{M}[W] = \mathscr{M}[X_0] \uplus \mathscr{M}[Y] \ \wedge \ IsSorted[W] \right) \right). \tag{54}$$

After Skolemizing Y to Y_0 the LHS of the implication becomes assumption, and the RHS becomes goal and uses the metavariable W^*:

$$\mathscr{M}[W^*] = \mathscr{M}[X_0] \uplus \mathscr{M}[Y_0] \ \wedge \ IsSorted[W^*]. \tag{55}$$

By **ST-1** (cover set) on X_0 :

Case 1: $X_0 = \langle\rangle$. By straightfoward proof the solution is $\{W^* \to Y_0\}$.

Case 2: $X_0 = a_0 \smile U_0$.

By **IR-5** (expand multiset) on $\mathscr{M}[a_0 \smile U_0]$ the goal becomes:

$$\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \ \wedge \ IsSorted[W^*]. \tag{56}$$

Alternative 2.1. By strategy **ST-2** (induction) which uses U_0 (smaller than X_0):

$$\mathscr{M}[Merge[U_0, Y_0]] = \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \ \wedge \ IsSorted[Merge[U_0, Y_0]]. \tag{57}$$

By rewriting using (57) the goal (56) becomes:

$$\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[Merge[U_0, Y_0]] \ \wedge \ IsSorted[W^*]. \tag{58}$$

Application of **ST-4** (pair multisets) on $\{\{a_0\}\}$ and $\mathscr{M}[Merge[U_0, Y_0]]$ and of **ST-3** (cascading) using (57) and (58) produces **Conjecture 5** which is used to generate the algorithm *Insert*:

$$\mathscr{M}[W^*] = \mathscr{M}[Insert[a, Merge[U_0, Y_0]]] \ \wedge \ IsSorted[W^*]. \tag{59}$$

By **IR-4** the solution is $\{W^* \to Insert[a_0, Merge[U_0, Y_0]]\}$ and the synthesized algorithm is:

Algorithm 9. Merge sorted lists using insert, version 1.

$$\underset{a,U,V}{\forall}\left(\begin{array}{c} Merge[\langle\rangle,V]=V \\ Merge[a\frown U,V]=Insert[a,Merge[U,V]] \end{array}\right)$$

This is of course not the most efficient algorithm because the induction is not used on both arguments (as it is done in the sequel, see below). A hint about inefficiency is that the property of U to be sorted is not used in the proof, but this has also a positive side: $Merge[U,\langle\rangle]$ is a sorting algorithm, essentially equivalent to *insert sort*.

Alternative 2.2. Applying strategy **ST-4** (pair multisets) to $\{\{a\}\}$ and $\mathscr{M}[V_0]$ and then **ST-3** (cascading) produces the same conjecture for *Insert*, and the goal becomes:

$$\mathscr{M}[W^*]=\mathscr{M}[U_0]\uplus\mathscr{M}[Insert[a,V_0]]\ \wedge\ IsSorted[W^*] \tag{60}$$

with the additional assumption: $IsSorted[Ins[a,V_0]]$. We can apply now strategy **ST-2** (induction) to the pair of multiset terms and construct the list U_1 which is sorted and whose multiset is equal to the union. Therefore the solution is $\{W^*\to U_1\}$ and the merging algorithm is:

Algorithm 10. Merge sorted lists using insert, version 2.

$$\underset{a,U,V}{\forall}\left(\begin{array}{c} Merge[\langle\rangle,V]=V \\ Merge[a\frown U,V]=Merge[U,Insert[a,V]] \end{array}\right)$$

This algorithm, although not optimal, is interesting because it is tail–recursive, and, since only the second argument needs to be sorted, it can also be used for sorting as $Merge[U,\langle\rangle]$, which is again *insert sort*.

Remark. If the proof continues from the goal (56) by applying strategies **ST-4** (pair multisets) to $\{\{a\}\}$ and $\mathscr{M}[U_0]$,and then **ST-3** (cascading), then induction cannot be applied to the resulting multiset pair ($\mathscr{M}[Insert[a_0,U_0]]$ and $\mathscr{M}[Y_0]$) because $Insert[a_0,U_0]$ is not smaller than the target constant $X_0=a_0\frown U_0$. The corresponding algorithm would have $Merge[a\frown U,V]=Merge[Insert[a,U],V]$ as the second clause, which is an infinite loop.

Alternative 2.3. The proof continues from goal (56) by applying **ST-1** (cover set) on Y_0 in a *nested* fashion: now we have a second target constant Y_0 and a second target goal obtained from (54):

$$(IsSorted[a_0\frown X_0]\wedge IsSorted[Y_0])\Longrightarrow\underset{W}{\exists}\,(\mathscr{M}[W]=\mathscr{M}[a_0\frown X_0]\uplus\mathscr{M}[Y_0]\ \wedge\ IsSorted[W]). \tag{61}$$

This is not a goal in the proof, but a pattern for generating new assumptions by induction, for ground terms smaller than Y_0, by strategy **ST-2**.

Case 2.3.1. $Y_0=\langle\rangle$: Similarly, the solution is $\{W^*\to a_0\frown U_0\}$.

Case 2.3.2. $Y_0=b_0\frown V_0$: By application of **IR-3** (reduce composite argument) to $IsSorted[Y_0]$:

$$IsSorted[b_0\frown V_0]\ \wedge\ b_0\le V_0\ \wedge\ IsSorted[V_0] \tag{62}$$

and the goal becomes:

$$\mathscr{M}[W^*]=\{\{a_0\}\}\uplus\mathscr{M}[U_0]\uplus\{\{b_0\}\}\uplus\mathscr{M}[V_0]\ \wedge\ IsSorted[W^*]. \tag{63}$$

When the rule **IR-8** (two constants) is applied, then one has:

Case 2.3.2.1. $a_0 \leq b_0$: A successful proof alternative proceeds by using first **IR-6** to replace $\{\{b_0\}\} \uplus \mathscr{M}[V_0]$ by $\mathscr{M}[b_0 \frown V_0]$, then by using **ST-4** (pair multisets) and **ST-2** (induction) on (54) to replace $\mathscr{M}[U_0] \uplus \mathscr{M}[b_0 \frown V_0]$ by $\mathscr{M}[Merge[U_0, b_0 \frown V_0]]$ (because U_0 is less than X_0, and as second argument of *Merge* any Y is allowed). After that, prefixing a_0 to this by **IR-6** (compress multiset) results in a sorted list by the current assumptions and the properties of the domain ordering.

Case 2.3.2.2. $b_0 < a_0$: A similar proof alternative succeeds, but here **ST-4** (pair multisets) is applied to $\mathscr{M}[a_0 \frown U_0]$ and V_0 and then **ST-2** (induction) can be used on the basis of the second target goal (61), because the second argument V_0 is less than Y_0 and the first argument is exactly as in the pattern. Finally the algorithm is the classical one:

Algorithm 11. Merge sorted lists, version 3.

$$\underset{a,b,U,V}{\forall} \left(\begin{array}{c} Merge[\langle\rangle, V] = V \\ Merge[a \frown U, \langle\rangle] = a \frown U \\ Merge[a \frown U, b \frown V] = \left\{ \begin{array}{ll} a \frown Merge[U, b \frown V], & \text{if } a \leq b \\ b \frown Merge[a \frown U, V], & \text{if } b < a \end{array} \right. \end{array} \right)$$

Because there are 4 multiset terms in the goal, strategy **ST-4** (pair multisets) generates many alternatives, which in turn lead to several algorithms, which only differ in the RHS of the last clause, but are less efficient than the one above. Some of them have interesting properties, for instance the one ending in $Merge[a \frown U, b \frown V] = Insert[a, Insert[b, Merge[U, V]]]$ will generate a sorted list even if the arguments are not sorted, while the one ending in $Merge[a \frown U, b \frown V] = Insert[b, Merge[a \frown U, V]]$ needs only the first argument to be sorted.

QED

6 Conclusions and Further Work

We demonstrate the possibility of automatic synthesis of complex algorithms on (possibly sorted) lists, using the notion of multiset. The proofs are more efficient than by using general resolution, because specific inference rules and strategies which are also taylored for synthesis proofs, notably for discovering concrete induction principles and for synthesizing needed auxiliary functions. 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 supported 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, as well as merging and inserting on lists and trees, and are a prerequisite for further work on synthesis of more complex algorithms for sorting and searching, including operations on several domains.

References

[1] W. D. Blizard. Multiset Theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989. doi:10.1305/ndjfl/1093634995.

[2] B. Buchberger. Algorithm Invention and Verification by Lazy Thinking. *Analele Universitatii din Timisoara, Seria Matematica - Informatica*, XLI:41–70, 2003.

[3] B. Buchberger and A. Craciun. Algorithm Synthesis by Lazy Thinking: Using Problem Schemes. In *Proceedings of SYNASC 2004*, pages 90–106, 2004.

[4] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema project: A progress report. In *Calculemus 2000*, pages 98–113. A.K. Peters, Natick, Massachusetts, 2000.

[5] 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. doi:10.6092/issn.1972-5787/4568.

[6] A. Bundy, L. Dixon, J. Gow, and J. Fleuriot. Constructing Induction Rules for Deductive Synthesis Proofs. *Electronic Notes Theoretical Computer Science*, 153:3–21, March 2006. doi:10.1016/j.entcs.2005.08.003.

[7] I. Dramnesc and T. Jebelean. Proof Techniques for Synthesis of Sorting Algorithms. In *SYNASC 2011*, pages 101–109. IEEE Computer Society, 2011. doi:10.1109/SYNASC.2011.23.

[8] I. Dramnesc and T. Jebelean. Automated synthesis of some algorithms on finite sets. In *SYNASC 2012*, pages 143 – 151. IEEE Computer Society, 2012. doi:10.1109/SYNASC.2012.43.

[9] I. Dramnesc and T. Jebelean. Theory Exploration in Theorema: Case Study on Lists. In *SACI 2012*, pages 421 – 426. IEEE Xplore, 2012. doi:10.1109/SACI.2012.6250041.

[10] I. Dramnesc and T. Jebelean. Theory Exploration of Sets represented as Monotone Lists. In *SISY 2014*, pages 163 – 168. IEEE Xplore, 2014. doi:10.1109/SISY.2014.6923579.

[11] I. Dramnesc and T. Jebelean. Synthesis of List Algorithms by Mechanical Proving. *Journal of Symbolic Computation*, 68:61–92, 2015. doi:10.1016/j.jsc.2014.09.030.

[12] I. Dramnesc and T. Jebelean. Case Studies on Algorithm Discovery from Proofs: The *Delete* Function on Lists and Binary Trees using Multisets. In *SISY 2019*. IEEE Xplore, 2019. (to appear).

[13] I. Dramnesc, T. Jebelean, and S. Stratulat. Theory Exploration of Binary Trees. In *SISY 2015*, pages 139 – 144. IEEE, 2015. doi:10.1109/SISY.2015.7325367.

[14] I. Dramnesc, T. Jebelean, and S. Stratulat. Proof-based Synthesis of Sorting Algorithms for Trees. In *LATA 2016*, pages 562–575. Springer, 2016. doi:10.1007/978-3-319-30000-9_43.

[15] 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. doi:10.1016/j.jsc.2018.04.002.

[16] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3 edition, 1998. doi:10.1137/1012065.

[17] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 1: Deductive Reasoning. Addison-Wesley, 1985. doi:10.2307/2275898.

[18] A. Radoaca. Properties of Multisets Compared to Sets. In *SYNASC 2015*, pages 187–188, 2015. doi:10.1109/SYNASC.2015.37.

[19] W. Windsteiger. Theorema 2.0: A System for Mathematical Theory Exploration. In *ICMS'2014*, volume 8592 of *LNCS*, pages 49–52, 2014. doi:10.1007/978-3-662-44199-2_9.