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

We experiment with the mechanical synthesis of list–sorting algorithms by extracting them from automatically generated proofs in the *Theorema* system. The basic principle consists in using the specification (input and output conditions) of the function to be synthesized for constructing a specific logical conjecture, and then proving this in a constructive way. The witnesses found during the proof, possible prefixed by certain conditions, define a set of rewrite rules which express the algorithm, and which can be directly executed in the *Theorema* system. The knowledge base includes properties of multisets and their relation to lists, in this way we can express naturally the fact that two lists have the same elements, and also introduce powerful inference rules and proof strategies based on multiset properties. The proofs are in natural style, using inference rules and strategies specific for the domains involved, but also general ones. In particular we present specific strategies for the construction of arbitrarily structured recursive algorithms by general Noetherian induction, as well as a systematic strategy for the "cascading" principle: the specification of needed auxiliary functions are automatically produced during the proof, and their synthesis is performed using the same method. These case studies are a part of our work on exploring the theory of these domains and demonstrating the automatic synthesis of complex algorithms like sorting and searching.

1 Introduction

We present a comprehensive case study in 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 two related papers [12, 13] we already investigated algorithms for deletion from lists and binary trees, as well as for insertion and merging.

We use of multisets because they allow to express naturally the idea that two lists have the same elements, and also that the collection of classical notions on sets (membership, union, intersection, etc.) can be used as a roadmap for the theory exploration.

We approach automated synthesis like described in our previous work – see e. g. [11, 16]. 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 [5], in which the inference rules and the logical formulae are presented in *natural style* – a 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 [6] and it was previously used by us [11, 16].

Submitted to:
FROM 2019

Related work. The theory of *multisets* is well studied in the literature, including computational formalizations (see e. g. [18], 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 [19]. Concerning *sorting* we are not aware of a systematic formalization of the theory of lists and trees using multisets. An interesting formalization in a previous version of *Theorema* [4], which includes the theory exploration and the synthesis of a sorting algorithm is presented 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. In our previous work we study proof–based algorithm synthesis in the theories of lists [9], sets [10] and binary trees [14] separately ([7], [8], [15], [11], [16].

Originality. In contrast to other investigations, the current study uses multisets in the theory development and in the entire process of algorithm synthesis. The automatically generated proofs are performed in the new version of the *Theorema* system [5, 20].

In contrast to our previous work on synthesis, 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 [16], 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.

Four novel inference rules and six novel strategies are introduced.

Moreover we use very systematically 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 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 [11], and in this paper we describe it in a more systematic manner as proof and we illustrate it on several example: all auxiliary algorithms are generated by cascading starting from sorting synthesis proofs.

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 $\underset{X}{\exists}$. Meta–variables are starred (e.g., T^*, T_1^*, Z^*) and Skolem constants have integer indices (e.g., X_0, X_1, a_0).

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

Elements 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 (denoted $\leq, <$) and between lists/multisets (denoted $\leq$), by requiring that all elements of the composite object observe the ordering relation[1]. Elements are denoted by lower case letters (a, b, x, y).

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

Finite **multisets** may contain the same elements several times (the multiplicity can be more than one). $\emptyset$ 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 [17]. Union is commutative and associative with unit $\emptyset$, these properties are used implicitly by the prover. $\mathscr{M}[X]$ denotes the multiset of elements of the list X.

A finite **list** is empty $\langle\rangle$ or of the form $a \smile U$, where $\smile$ is the operation of prepending an element to a list (analogous to *cons* of *Lisp*). The multiset of a list observes:

Property 1. $\underset{a,V}{\forall} \begin{pmatrix} \mathscr{M}[\langle\rangle] = \emptyset \\ \mathscr{M}[a \smile V] = \{\{a\}\} \uplus \mathscr{M}[V] \end{pmatrix}$

Sorted list are defined by:

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

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 main 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 a specific *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) introduced for existential assumptions and universal goals, as well as *metavariables* (denoted with star power like T^*)introduced for existential goals. Few inference rules and strategies are similar to the ones described in [11–13, 16], but most are novel ones, namely the inference rules: **IR-3**, **IR-4**, **IR-5**, and **IR-12**, as well as the strategies: **ST-2**, **ST-3**, **ST-4**, **ST-5**, **ST-6**, and **ST-8**.

2.2.1 Inference Rules

IR-1: *Split conjunctive assumptions.* A conjunctive assumption is split into its conjuncts.
IR-2: *Assumed subgoal.* In a conjuctive goal, delete the part which is already an assumption.
IR-3: *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. This rule is akin to unit resolution and realizes for instance Modus Ponens and Modus Tollens.
IR-4: *Backward inference.* Transform the goal using some assumption or a specific logical principle. If a ground atomic assumption matches a part of an 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. (The rule **IR-2** is a special case of this when there is no variable involved.)

A specific logical principle is used for backward inference on goals containing metavariables, namely the fact that a formula having the structure $\exists_x P[x]$ is a logical consequence of the formula $\exists_x P[f[x]]$, where P is an arbitrary predicate (in fact a formula) and f an arbitrary function (in fact a term).

IR-5: *Rewrite by equality.* Uses an equality assumption to transforms part of the goal or of an assumption. For instance, in the goal, replace a ground subterm by another which contains more information, (like $\mathscr{M}[U_0]$ by $\mathscr{M}[Sort[U_0]]$), or simplify a term containing a metavariable (like replace $Sort[W^*]$ by W^*) in order ease the constraint.

IR-6: *Reduce composite argument.* This rule uses the current assumptions to transform parts of the goal or of the assumptions into atoms whose arguments contain no function symbols.
Example: $a_0 \leq Conc[U_0, Y_0]$ becomes $a_0 \leq U_0 \wedge a_0 \leq Y_0$.

IR-7: *Solve metavariable.* When the goal is $\mathscr{M}[X^*] = \mathscr{M}[\mathscr{T}]$ for a ground term $\mathscr{T}$, infer $X^* = \mathscr{T}$.

IR-8: *Expand multiset.* In the goal, a multiset term with a composite argument is expanded by equality into several multiset terms. Typically this is 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 more information, for instance by applying induction.
Example: The goal $\mathscr{M}[T^*] = \mathscr{M}[a_0 \frown U_0] \uplus \mathscr{M}[V_0]$, becomes $\mathscr{M}[T^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \uplus \mathscr{M}[V_0]$.

IR-9: *Compress multiset.* This is the dual of the previous rule, and it is most often applied using $\frown$.
Example: the goal is $\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[W_1]$ becomes $\mathscr{M}[W^*] = \mathscr{M}[a_0 \frown W_1]$.

IR-10: *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 1: if a part of the goal is $b \leq W_1$ and an assumption is: $\mathscr{M}[W_1] = \{\{a_0\}\} \uplus \mathscr{M}[W_0]$, then the respective part of the goal is transformed into $b \leq a_0 \wedge b \leq W_0$.
Example 2: the subgoal $a^* \leq Sort[W^*]$ is replaced by $a^* \leq W^*$.

IR-11: *Simple goal conditional assumption.* When the proof fails and the current goal is ground and contains only simple elements (not lists), then the proof stops and the unproved goal will become a condition in the synthesized algorithm, as explained below at strategy **ST-7**.

IR-12: *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 $b_0 < a_0$.

2.2.2 Strategies

ST-1: *Quantified variables.* Main universal variables in goal (but *not* the main existential variables in assumptions) become Skolem constants. Main existential variables in goal (but *not* the main universal variables in assumptions) become metavariables.

ST-2: *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[2] which represent the domain of the main universal variable [16]. We project this concept on Skolem constants: first the

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

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*. (During the proof some Skolem constants[3] may be further refined by such a cover–set decomposition.) At the beginning of the proof, the prover chooses a covers set (typically the one suggested by the recursive definition of the domain), and starts a proof branch for each ground term ("proof by cases"). On each proof branch the input conditions of the functions 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. Some branches will split further by introducing conditions or by refining the cover sets of some Skolem constants present in the proof as described above. Finally the algorithm will be generated as a set of [conditional] equalities: the terms of the cover set will be the arguments ("patterns") on the LHS of the equalities, and the corresponding witnesses will be the RHS of these, after replacing back the Skolem constants by variables.

The strategy is applied similarly to a metavariable from the goal, 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.

ST-3: *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 $\mathscr{M}[a_0 \cup U_0]$ is a cover–set term for the target constant $\mathscr{M}[X_0]$ then U_0 is smaller than X_0.

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 $P[t]$ may be used as an assumption. The prover applies this by inserting the target function directly instead of the existential quantifed variable.

Example: The target function is $F[X,Y]$, the target goal $P[X_0]$ is $\underset{Y}{\forall}\,(I[X_0] \implies \underset{Z}{\exists}\, O[X_0,Y,Z])$, ($X_0$ is the target constant), the current cover–set term is t' (note that t' is an alternative representation of X_0), and t is smaller than X_0 in the well–founded ordering. The instance $P[t]$ of the target goal is $\underset{Y}{\forall}\,(I[t,Y] \implies \underset{Z}{\exists}\, O[t,Y,Z])$, The prover adds the assumption $\underset{Y}{\forall}\,(I[t,Y] \implies O[t,Y,F[t,Y]])$ 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.

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 used as an assumption for reducing the current goal (see reduction of goal (10) in **Proof 1**).

ST-4: *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

[3] In fact also ground terms can be subject to cover–set decomposition, and this needs in the synthesized algorithm, because the conditional or the pattern matching branching, also an *assignment* programming instruction.

variables $y, y', \ldots$, and the conjecture has the structure[4]:

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

$P[x,x',\ldots]$ 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. The goal and the assumptions will change by using this property by using the inference rules of the prover, however in certain special (and typical) situations the changes can be applied directly as described in the sequel:

When a subgoal is $\mathscr{M}[X^*] = \mathscr{M}[t] \uplus \mathscr{M}[t'] \uplus \ldots$, (where $t, t', \ldots$ are ground — may contain Skolem constants), then let $\mathscr{M}[t_1], \mathscr{M}[t_2]$ be a pair such that the type of t_2 is a list. The conjecture is[5] $\underset{x}{\forall}\underset{X}{\forall} (P[x,X] \Longrightarrow \underset{Y}{\exists} Q[x,X,Y])$. If t_1 is $\{\{a\}\}$ (a domain element), then x represents this element, while if t_1 is of list type, then so is also x. Note that for lists the whole terms (t_1,t_2) correspond to the universal variables in the conjecture, and not the individual Skolem constants.The function $f[x,X]$ generated by cascading has the property $\underset{x}{\forall}\underset{X}{\forall} (P[x,X] \Longrightarrow Q[x,X,F[x,X]])$ and in the corresponding subgoal the pair $\mathscr{M}[t_1], \mathscr{M}[t_2]$ is replaced by $\mathscr{M}[F[a,t_2]]$ (if t_1 is $\{\{a\}\}$) or by $\mathscr{M}[F[t_1,t_2]]$ (if t_1 is a list).

ST-5: *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 this pair a conjecture is created as described at strategy **ST-4**, and its result is considered to be this goal (as opposite to "true" when the proof succeeds, or "false" when it fails). from which a multiset term which equals the union of the pairs can be constructed in one of the following ways:

(i) the corresponding function is already known, then the proof works by predicate logic.

(ii) induction can be applied (if the target function is binary);

(iii) a separate synthesis proof of the conjecture is necessary – see **ST-4** (cascading).

ST-6: *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 $a < X_2$. For this a certain conjecture is produced, which is proven either because the properties of the appropriate functions are among the assumptions, or by cascading. After cascading the defining property of the two functions is added to the assumptions in order to be reused if necessary. Furthermore this property is instantiated with the terms which triggered the split, namely $\{\{a\}\}$ and X and in the goal $\mathscr{M}[X]$ is replaced by $\mathscr{M}[X_1] \uplus \mathscr{M}[X_2]$ reordering the union such that the sorting is easier.

ST-7: *Conditional branches.* When the proof finishes by inference rule **IR-11** (simple goal conditional assumption), the prover creates a parallel branch in which the negation of the respective conditional

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

[5] By convention, here x represents any kind of objects: domain elements or lists, while X, Y represent lists.

assumption is assumed. In this way the prover discovers an useful disjunction to be used in *proof by cases*, and which corresponds to a conditional in the synthesized algorithm.
ST-8: *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.

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.*
By **IR-1** (quantified variables): Skolemize universal X 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 use metavariable V^* for the existential V :

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

Two alternatives are pursued, by applying strategy **ST-2** (cover set) to the metavariable V^* or to the Skolem constant X_0 :

Alternative 1: Apply **ST-2** (cover set) to V^* with the cover set determined by the domain definition: $\{\langle\rangle,\ a^* \frown U^*\}$

Case 1: $\langle\rangle$: The goal (4) becomes:

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

By **IR-2** (assumed subgoal) using **Definition 1** the goal (5) becomes:

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

By **ST-2** (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 2: $\{a^* \frown U^*\}$: 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-6** (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-4** (backward inference) U^* is replaced by $Sort[W^*]$ and the goal becomes:

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

and the intermediate solution for V^* is $a^* \backsim Sort[W^*]$. Since $a^* \backsim Sort[W^*]$ 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 **ST-3** (induction) the target goal 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:

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

The first conjunct is used by **IR-10** (use equivalence) to reduce the last goal to:

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

Apply **ST-4** (cascading) to this goal and generate the conjecture:
Conjecture 4. $\underset{X}{\forall}\,(X \neq \langle\rangle \Longrightarrow \underset{a}{\exists}\,\underset{U}{\exists}\,(\mathscr{M}[a \backsim U] = \mathscr{M}[X] \wedge a \leq U))$.
The proof **Proof 3** synthesizes the functions $min[X]$ and $Trim[X]$ which split a list into its minimum and the rest. By **ST-4** (cascading) the new assumption is:

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

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 $min[X_0] \backsim Sort[Trim[X_0]]$. The algorithm is:
Algorithm 1. Min-Sort.

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

Alternative 2: Apply **ST-2**: choose for X_0 the cover set suggested by the definition of lists: $\{\langle\rangle,\ a_0 \backsim U_0\}$, and start the corresponding two branches:

Case 1: $\langle\rangle$ is trivial. The solution is $\{V^* \to \langle\rangle\}$.

Case 2: $a_0 \backsim U_0$:
Goal:

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

By **IR-8** (expand multiset) using **Property 1** the goal becomes:

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

Two alternatives are pursued, depending on how this goal is treated.

Alternative 1: Strategy **ST-3** (induction) uses U_0 (smaller than X_0) to produce the assumption:

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

By **IR-1** (split conjunction), assumption (17) becomes:

$$\mathscr{M}[Sort[U_0]] = \mathscr{M}[U_0] \tag{18}$$

$$IsSorted[Sort[U_0]] \tag{19}$$

By **IR-5** (rewrite by equality) using (18) transform (16) into:

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

Apply **ST-5** (pair multisets) to $\{\{a\}\}$ and $\mathscr{M}[Sort[U_0]]$ using (19) and (20) to produce 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 proof of this conjecture is **Proof 6** and it produces the function $Insert[a,X]$ which inserts an element in a sorted list, keeping it sorted. Using the principles from **ST-4** (cascading) the new assumption is:

$$\underset{a}{\forall}\underset{X}{\forall}(IsSorted[X] \implies (\mathscr{M}[Insert[a,X]] = \{\{a\}\} \uplus \mathscr{M}[X] \wedge IsSorted[Insert[a,X]])). \tag{21}$$

and the goal (20) becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[Insert[a_0, Sort[U_0]]] \wedge IsSorted[V^*]. \tag{22}$$

By **IR-7** (solve metavariable) the solution for V^* is $Insert[a_0, Sort[U_0]]$ and the goal reduces to the assumption 19, thus the proof succeeds and 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: 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 apply strategy **ST-6** (split) 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 $SmallEq[a,X]$ and $Bigger[a,X]$ which split the list X into two lists having elements which are smaller, respectively bigger than a.

By strategy **ST-4** (cascading) the new assumption is:

$$\underset{a}{\forall}\underset{X}{\forall}(\mathscr{M}[X] = \mathscr{M}[SmallEq[a,X]] \uplus \mathscr{M}[Bigger[a,X]] \wedge SmallEq[a,X] \leq a \wedge a < Bigger[a,X]). \tag{23}$$

By strategy **ST-6** (split) this is instantiated with a_0 and U_0 to produce:

$$\begin{aligned} \mathscr{M}[U_0] = \mathscr{M}[SmallEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \wedge \\ SmallEq[a_0,U_0] \leq a_0 \wedge a_0 < Bigger[a_0,U_0]). \end{aligned} \tag{24}$$

and the goal (16) is transformed into:

$$\mathscr{M}[V^*] = \mathscr{M}[SmallEq[a_0,U_0]] \uplus \{\{a_0\}\} \uplus \mathscr{M}[Bigger[a_0,U_0]] \wedge IsSorted[V^*]. \tag{25}$$

The conjunction (24) is decomposed into three assumptions by **IR-1**:

$$\mathscr{M}[U_0] = \mathscr{M}[SmallEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \tag{26}$$

$$SmallEq[a_0,U_0] \leq a_0 \tag{27}$$

$$a_0 < Bigger[a_0,U_0]). \tag{28}$$

Because (26) neither of $SmallEq[a_0,U_0]$ 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-3** (induction) can be applied to both of them, producing the assumptions:

$$\mathscr{M}[SmallEq[a_0,U_0]] = \mathscr{M}[Sort[SmallEq[a_0,U_0]]] \tag{29}$$

$$IsSorted[Sort[SmallEq[a_0,U_0]]] \tag{30}$$

$$\mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[Sort[Bigger[a_0,U_0]]] \tag{31}$$

$$IsSorted[Sort[Bigger[a_0,U_0]]] \tag{32}$$

By **IR-5** using (29) and (31) replace in goal (16) the corresponding subterms to obtain:

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[SmallEq[a_0,U_0]]] \uplus \{\{a_0\}\} \uplus \mathscr{M}[Sort[Bigger[a_0,U_0]]] \wedge IsSorted[V^*]. \tag{33}$$

Apply strategy **ST-5** (pair multisets) to $\{\{a_0\}\}$ and $\mathscr{M}[Sort[Bigger[a_0,U_0]]]$ and produce:
Conjecture 7. $\underset{a}{\forall}\underset{X}{\forall}((a \leq X \wedge IsSorted[X]) \Longrightarrow \underset{V}{\exists}(\mathscr{M}[V] = \{\{a\}\} \uplus \mathscr{M}[X] \wedge IsSorted[V]))$.
This is solved by the existing function *cons* and the new goal is:

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[SmallEq[a_0,U_0]]] \uplus \mathscr{M}[a_0 \backsim Sort[Bigger[a_0,U_0]]] \wedge IsSorted[V^*]. \tag{34}$$

By forward inference using the current assumptions and the properties of inequality we obtain: $Sort[SmallEq[a_0,U_0]] \leq a_0 < Sort[Bigger[a_0,U_0]]]$, $IsSorted[a_0 \backsim Sort[Bigger[a_0,U_0]]]$ and $Sort[SmallEq[a_0,U_0]] \leq a_0 \backsim Sort[Bigger[a_0,U_0]]$.
Pair multisets $\mathscr{M}[Sort[SmallEq[a_0,U_0]]]$ and $\mathscr{M}[a_0 \backsim Sort[Bigger[a_0,U_0]]]$ to obtain:
Conjecture 8. $\underset{X}{\forall}\underset{Y}{\forall}((X \leq Y \wedge IsSorted[X] \wedge IsSorted[Y]) \Longrightarrow \underset{V}{\exists}(\mathscr{M}[V] = \mathscr{M}[X] \uplus \mathscr{M}[Y] \wedge IsSorted[V]))$.
The proof **Proof 7** generates the algorithm *Conc* for concatenation which puts two lists together, and if the conditions are like above, the result is sorted. The new goal is:

$$\mathscr{M}[V^*] = \mathscr{M}[Conc[Sort[SmallEq[a_0,U_0]]], a_0 \backsim Sort[Bigger[a_0,U_0]]] \wedge IsSorted[V^*]. \tag{35}$$

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

Algorithm 3. Quick-Sort.

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

QED

Another approach is to consider a covers set corresponding to the *divide–and–conquer* principle: $\{\langle\rangle,\ a \backsim \langle\rangle,\ Conc[U,V]\}$ (where U,V are nonempty). Here *Conc* is used as a *pattern matching* construct, which can be used on the LHS of a rewrite rule, and it comes together with a simple splitting function, which gives two nonempty lists from a list with 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:

$$\mathscr{M}[V^*] = \mathscr{M}[X_0] \wedge IsSorted[V^*]. \tag{36}$$

In all alternatives the cases $\langle\rangle$ and $a \backsim \langle\rangle$ are straightforward.

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

Alternative 2: Application of the cover–set strategy to X_0.

Case 1: $\langle\rangle$ straightforward.

Case 2: $a_0 \backsim \langle\rangle$ straightforward.

Case 3: $Conc[U_1,U_2]$: After splitting the multiset the goal becomes:

$$\mathscr{M}[V^*] = \mathscr{M}[U_1] \uplus \mathscr{M}[U_2] \wedge IsSorted[V^*]. \tag{37}$$

After induction[6] (we do not list the obvious assumptions):

$$\mathscr{M}[V^*] = \mathscr{M}[Sort[U_1]] \uplus \mathscr{M}[Sort[U_2]] \wedge IsSorted[V^*]. \tag{38}$$

By pairing the two multisets we have the conjecture:

Conjecture 9.

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

The proofs in subsection 5.3 synthesize in several versions of the algorithm *Merge* which combines two sorted lists into a sorted one. The corresponding sorting algorithm is:

Algorithm 4. Merge Sort.

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

QED

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

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:

$$X_0 \neq \langle\rangle \tag{39}$$

and prove:

$$\underset{y}{\exists}\,\underset{Y}{\exists}\,(\mathscr{M}[X_0] = \mathscr{M}[Y] \uplus \{\{y\}\} \wedge y \leq X_0). \tag{40}$$

By **ST-1** the goal is:

$$\mathscr{M}[X_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq X_0. \tag{41}$$

Apply **ST-2** (cover set) on X_0, using only $a_0 \frown U_0$ because (39).

Case 1: $X_0 : a_0 \frown U_0$:

The goal is:

$$\mathscr{M}[a_0 \frown U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq a_0 \frown U_0. \tag{42}$$

Apply **IR-6** and the goal becomes:

$$\mathscr{M}[a_0 \frown U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq a_0 \wedge y^* \leq U_0. \tag{43}$$

Apply **ST-4** (cascading) to generate the conjecture:

Conjecture 10. $\underset{X}{\forall}\,\underset{a}{\forall}\,\underset{y}{\exists}\,\underset{Y}{\exists}\,(\mathscr{M}[a \frown X] = \mathscr{M}[Y] \uplus \{\{y\}\} \wedge y \leq a \wedge y \leq X)$.

Proof 4 synthesizes *minAux* and *TrimAux* which have the property:

$$\underset{X}{\forall}\,\underset{a}{\forall}\,(\mathscr{M}[a \frown X] = \mathscr{M}[TrimAux[a,X]] \uplus \{\{minAux[a,X]\}\} \wedge minAux[a,X] \leq a \wedge minAux[a,X] \leq X) \tag{44}$$

and which solvethe goal (43) using the witnesses $\{Y^* \to TrimAux[a_0,U_0],\ y^* \to minAux[a_0,U_0]\}$.

QED

We prove now **Conjecture 10**.
Proof 4: *Min and Trim auxiliary.*
By **ST-1** the goal is:

$$\mathscr{M}[a_0 \frown X_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq a_0 \wedge y^* \leq X_0 \tag{45}$$

Apply **ST-2** (cover set) on X_0.

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

Case 2: $X_0 : b_0 \frown U_0$ Prove:

$$\mathscr{M}[a_0 \frown (b_0 \frown U_0)] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq a_0 \wedge y^* \leq b_0 \frown U_0 \tag{46}$$

Apply **IR-8** (expand multisets) and **IR-6** (reduce composite argument) and the goal becomes:

$$\{\{a_0\}\} \uplus \{\{b_0\}\} \uplus \mathscr{M}[U_0] = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge y^* \leq a_0 \wedge y^* \leq b_0 \wedge y^* \leq U_0 \tag{47}$$

Two cases for domain element constants are generated by rule **IR-12** (two constants):

Case 1:

$$a_0 \leq b_0 \tag{48}$$

Apply **ST-3** (induction) on U_0, a_0 in (45) and add the assumption:

$$\begin{aligned} \mathscr{M}[U_0] \uplus \{\{a_0\}\} = \mathscr{M}[TrimAux[a_0,U_0]] \uplus \{\{minAux[a_0,U_0]\}\} \wedge \\ minAux[a_0,U_0] \leq a_0 \wedge minAux[a_0,U_0] \leq U_0 \end{aligned} \tag{49}$$

Apply **IR-5** (rewrite by equality) in (47) using (49) and the goal becomes:

$$\begin{aligned} \mathscr{M}[TrimAux[a_0,U_0]] \uplus \{\{minAux[a_0,U_0]\}\} \uplus \{\{b_0\}\} = \mathscr{M}[Y^*] \uplus \{\{y^*\}\} \wedge \\ y^* \leq a_0 \wedge y^* \leq b_0 \wedge y^* \leq U_0 \end{aligned} \tag{50}$$

Split goal equation by strategy **ST-8** to transform the goal into:

$$\begin{aligned} \mathscr{M}[TrimAux[a_0,U_0]] \uplus \{\{b_0\}\} = \mathscr{M}[Y^*] \wedge \\ \{\{minAux[a_0,U_0]\}\} = \{\{y^*\}\} \wedge y^* \leq a_0 \wedge y^* \leq b_0 \wedge y^* \leq U_0 \end{aligned} \tag{51}$$

Apply **IR-7** to obtain solutions:
$\{y^* \to minAux[a_0,U_0], Y^* \to b_0 \backsim TrimAux[a_0,U_0]\}$ and the remaining goal is:

$$minAux[a_0,U_0] \leq a_0 \wedge minAux[a_0,U_0] \leq b_0 \wedge minAux[a_0,U_0] \leq U_0 \tag{52}$$

Apply **IR-2** using (49) and the remaining goal is:

$$minAux[a_0,U_0] \leq b_0 \tag{53}$$

From (49) and (48) by transitivity goal (53) is proven.

Case 2:

$$b_0 < a_0 \tag{54}$$

The proof proceeds similarly (apply (induction) on U_0, b_0 in (45)) and the obtained solutions are: $\{y^* \to minAux[b_0,U_0], Y^* \to a_0 \backsim TrimAux[b_0,U_0]\}$.

QED

The extracted algorithms from the proofs are:

Algorithm 5. *Min.*

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

Algorithm 6. *Trim.*

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

4.2 Split into smaller/bigger elements.

We need functions $SmallEq[a,X]$ 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.*
Apply **ST-1** skolemize a to a_0 and X to X_0 (target constant), metavariables V^*, W^*

$$\mathscr{M}[X_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \wedge V^* \leq a_0 \wedge a_0 < W^*. \tag{55}$$

Apply strategy **ST-2** (cover set) $\{\langle\rangle, a_1 \frown U_0\}$

Case 1: $\langle\rangle$ is trivial. The solutions are: $\{V^* \to \langle\rangle, W^* \to \langle\rangle\}$

Case 2: $a_1 \frown U_0$

$$\mathscr{M}[a_1 \frown U_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \wedge V^* \leq a_0 \wedge a_0 < W^*. \tag{56}$$

Apply **IR-8** (expand multiset)

$$\{\{a_1\}\} \uplus \mathscr{M}[U_0] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \wedge V^* \leq a_0 \wedge a_0 < W^*. \tag{57}$$

Apply **ST-3** (induction) on U_0 (smaller than X_0) and add the new assumption:

$$\mathscr{M}[U_0] = \mathscr{M}[SmallEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \wedge SmallEq[a_0,U_0] \leq a_0 \wedge a_0 < Bigger[a_0,U_0]. \tag{58}$$

Apply **IR-1** (split conjunction)

$$\mathscr{M}[U_0] = \mathscr{M}[SmallEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] \tag{59}$$

$$SmallEq[a_0,U_0] \leq a_0 \wedge a_0 < Bigger[a_0,U_0] \tag{60}$$

Apply **IR-5** (replace by equality in goal)

$$\{\{a_1\}\} \uplus \mathscr{M}[SmallEq[a_0,U_0]] \uplus \mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[V^*] \uplus \mathscr{M}[W^*] \wedge V^* \leq a_0 \wedge a_0 < W^*. \tag{61}$$

By inference rule **IR-12** (two constants) issue cases:

Case 1:

$$a_1 \leq a_0 \tag{62}$$

Apply strategy **ST-8** (split goal equation) to change the goal:

$$\{\{a_1\}\} \uplus \mathscr{M}[SmallEq[a_0,U_0]] = \mathscr{M}[V^*] \wedge V^* \leq a_0 \tag{63}$$

$$\mathscr{M}[Bigger[a_0,U_0]] = \mathscr{M}[W^*] \wedge a_0 < W^*. \tag{64}$$

Apply **IR-9** in (63), **IR-7** in both (63) and (64), the obtained solutions are:
$\{V^* \to a_1 \frown SmallEq[a_0,U_0], W^* \to Bigger[a_0,U_0]\}$ and the remaining goal is true by applying **IR-2** using (60).

Case 2:

$$a_0 < a_1 \tag{65}$$

Similarly, the obtained solutions are: $\{V^* \to SmallEq[a_0,U_0], W^* \to a_1 \frown Bigger[a_0,U_0]\}$.

QED

The extracted algorithms from the proof are:

Algorithm 7. *SmallEq*

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

Algorithm 8. *Bigger*

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

5 Merging

5.1 Adding an element to a sorted list

We prove **Conjecture 5**.

Proof 6: *Insert.*

By **IR-1** (quantified variables) the goal becomes:

$$IsSorted[X_0] \Longrightarrow (\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[X_0] \wedge IsSorted[V^*]) \tag{66}$$

Apply **ST-2** (cover set) on X_0 :

Case 1: $\langle\rangle$ is trivial. The solution is $\{V^* \to a\backsim\langle\rangle\}$.

Case 2: $b_0\backsim U_0$:

Assume:

$$IsSorted[b_0\backsim U_0] \tag{67}$$

which is expanded by applying **IR-6** using Definition 1 in:

$$b_0 \leq U_0 \wedge IsSorted[U_0] \tag{68}$$

Prove:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[b_0\backsim U_0] \wedge IsSorted[V^*] \tag{69}$$

Apply **IR-8** and the goal becomes:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \{\{b_0\}\} \uplus \mathscr{M}[U_0] \wedge IsSorted[V^*] \tag{70}$$

Apply twice **IR-9** and the new goal is:

$$\mathscr{M}[V^*] = \mathscr{M}[a_0\backsim(b_0\backsim U_0)] \wedge IsSorted[V^*] \tag{71}$$

Apply **IR-7**, obtain substitution $\{V^* \to a_0\backsim(b_0\backsim U_0)\}$ and prove:

$$IsSorted[a_0\backsim(b_0\backsim U_0)] \tag{72}$$

Apply **IR-6** and the goal becomes:

$$a_0 \leq b_0 \wedge b_0 \leq U_0 \wedge IsSorted[U_0] \tag{73}$$

Apply **IR-2** using (68) and the remaining goal is:

$$a_0 \leq b_0 \tag{74}$$

Which, according to **IR-11** becomes the conditional assumption on this branch.

Apply **ST-7** using (74) to generate another branch in the proof with the assumption:

$$b_0 < a_0 \tag{75}$$

Prove:

$$\mathscr{M}[V^*] = \{\{b_0\}\} \uplus \{\{a_0\}\} \uplus \mathscr{M}[U_0] \wedge IsSorted[V^*] \tag{76}$$

Apply **ST-3** (induction) on U_0 (smaller than X_0) and add the assumptions:

$$\mathscr{M}[Insert[a_0, U_0]] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \tag{77}$$

$$IsSorted[Insert[a_0, U_0]] \tag{78}$$

Apply **IR-5** using (77) and the goal (76) becomes:

$$\mathscr{M}[V^*] = \{\{b_0\}\} \uplus \mathscr{M}[Insert[a_0, U_0]] \wedge IsSorted[V^*] \tag{79}$$

Apply **IR-9** and the goal is:

$$\mathscr{M}[V^*] = \mathscr{M}[b_0 \backsim Insert[a_0, U_0]] \wedge IsSorted[V^*] \tag{80}$$

Apply **IR-7**, the substitution is $\{V^* \rightarrow b_0 \backsim Insert[a_0, U_0]\}$ with the remaining goal:

$$IsSorted[b_0 \backsim Insert[a_0, U_0]] \tag{81}$$

which is proven by applying **IR-6** and **IR-2** using (78),(68) and (75)

QED

From this proof the following algorithm is extracted:

Algorithm 9. *Insert an element in a sorted list.*

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

5.2 Merging two sorted lists knowing $X \leq Y$

We need to prove **Conjecture 8**.
Proof 7: *Concatenation.*

By **IR-1** (quantified variables) the goal becomes:

$$(X_0 \leq Y_0 \wedge IsSorted[X_0] \wedge IsSorted[Y_0]) \Longrightarrow (\mathscr{M}[V^*] = \mathscr{M}[X_0] \uplus \mathscr{M}[Y_0] \wedge IsSorted[V^*]) \tag{82}$$

Apply **ST-2** (cover set) on X_0 :

Case 1: $\langle\rangle$ is trivial. The solution is $\{V^* \to Y_0\}$.

Case 2: $\{a_0 \backsim U_0\}$:
Assume:

$$a_0 \backsim U_0 \leq Y_0 \wedge IsSorted[a_0 \backsim U_0] \wedge IsSorted[Y_0] \tag{83}$$

$$a_0 \leq Y_0 \wedge U_0 \leq Y_0 \wedge a_0 \leq U_0 \wedge IsSorted[U_0] \tag{84}$$

Prove:

$$\mathscr{M}[V^*] = \mathscr{M}[a_0 \backsim U_0] \uplus \mathscr{M}[Y_0] \wedge IsSorted[V^*] \tag{85}$$

Apply **IR-8** (expand multiset) and the new goal is:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \wedge IsSorted[V^*] \tag{86}$$

Apply **ST-3** (induction on U_0 smaller than X_0) and add the assumptions:

$$\mathscr{M}[Conc[U_0, Y_0]] = \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \tag{87}$$

$$IsSorted[Conc[U_0, Y_0]] \tag{88}$$

Apply **IR-5** (rewrite by equality) using (87) and the goal (86) becomes:

$$\mathscr{M}[V^*] = \{\{a_0\}\} \uplus \mathscr{M}[Conc[U_0, Y_0]] \wedge IsSorted[V^*] \tag{89}$$

Apply **IR-9** and prove:

$$\mathscr{M}[V^*] = \mathscr{M}[a_0 \backsim Conc[U_0, Y_0]] \wedge IsSorted[V^*] \tag{90}$$

Apply **IR-7**, the obtained substitution is $\{V^* \to a_0 \backsim Conc[U_0, Y_0]\}$ and the remaining goal is:

$$IsSorted[a_0 \backsim Conc[U_0, Y_0]] \tag{91}$$

Which is proven by applying **IR-6** and **IR-2** using (88), (84).

QED
The extracted algorithm from the proof is:

Algorithm 10. Concatenation

$$\underset{a,U,Y}{\forall} \begin{pmatrix} Conc[\langle\rangle, Y] = Y \\ Conc[a \backsim U, Y] = a \backsim Conc[U, Y] \end{pmatrix}$$

5.3 Merging two sorted lists in a sorted list

We prove **Conjecture 9**.
Proof 8: *Merge.*

By **IR-1** (quantified variables) the goal becomes:

$$IsSorted[X_0] \wedge IsSorted[Y_0]) \Longrightarrow (\mathscr{M}[W^*] = \mathscr{M}[X_0] \uplus \mathscr{M}[Y_0] \wedge IsSorted[W^*]) \tag{92}$$

Apply **ST-2** (cover set) on X_0 :

Case 1: $\langle\rangle$: Similar as in the previous proof the solution is $\{W^* \to Y_0\}$.

Case 2: $a_0 \backsim U_0$:
The goal is similar with (85) and is expanded in

$$\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \wedge IsSorted[W^*] \tag{93}$$

Alternative 1: Apply strategy **ST-3** which uses U_0 (smaller than X_0) to add the assumptions:

$$\mathscr{M}[Merge[U_0, Y_0]] = \mathscr{M}[U_0] \uplus \mathscr{M}[Y_0] \tag{94}$$

$$IsSorted[Merge[U_0, Y_0]] \tag{95}$$

Apply **IR-5** using (94) and goal (93) becomes:

$$\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[Merge[U_0, Y_0]] \wedge IsSorted[W^*] \tag{96}$$

Apply **ST-4** (cascading) to $\{\{a_0\}\}$ and $\mathscr{M}[Merge[U_0, Y_0]]$ using (95) and (96) to produce **Conjecture 5**. The proof of this conjecture is **Proof 6**. Using the principles from **ST-4** the new assumption is:

$$\underset{a}{\forall}\underset{X}{\forall}\Big(IsSorted[X] \Longrightarrow (\mathscr{M}[Insert[a,X]] = \{\{a\}\} \uplus \mathscr{M}[X] \wedge IsSorted[Insert[a,X]])\Big) \tag{97}$$

and goal (96) becomes:

$$\mathscr{M}[W^*] = \mathscr{M}[Insert[a, Merge[U_0, Y_0]]] \wedge IsSorted[W^*] \tag{98}$$

By **IR-7** obtain $\{W^* \to Insert[a_0, Merge[U_0, Y_0]]\}$ and the remaining goal is:

$$IsSorted[Insert[a, Merge[U_0, Y_0]]] \tag{99}$$

which by implicit properties reduces to assumption (95).
The extracted algorithm from the proof is:
Algorithm 11. *Merge sorted lists using insert, version 1.*

$$\underset{a,U,V}{\forall}\begin{pmatrix} Merge[\langle\rangle, V] = V \\ Merge[a \backsim U, V] = Insert[a, Merge[U, V]] \end{pmatrix}$$

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: the proof may continue from the goal (93) by applying strategies **ST-5** (pair multisets) and **ST-4** (cascading) to the multiset terms $\{\{a\}\}$ and $\mathscr{M}[V_0]$. First the same cascading conjecture will be produced for the same insertion function, and the goal becomes:

$$\mathscr{M}[W^*] = \mathscr{M}[U_0] \uplus \mathscr{M}[Ins[a,V_0]] \wedge IsSorted[W^*] \tag{100}$$

with the additional assumption: $IsSorted[Ins[a,V_0]]$. We can apply now strategy **ST-3** (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 12. *Merge sorted lists using insert, version 2.*

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

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. One may say that the proof may continue from the goal (93) by applying strategies **ST-5** (pair multisets) and **ST-4** (cascading) to the multiset terms $\{\{a\}\}$ and $\mathscr{M}[U_0]$, but would result in an algorithm which generates an infinite loop

Algorithm 13. *Merge sorted lists using insert (infinite loop).*

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

This algorithm is not generated by our prover because $Insert[a_0, U_0]$ is not smaller than $a_0 \backsim U_0$ and is not accepted as assumption at the induction step.

Alternative 3: the proof may continue from the goal (93) by applying **ST-2** (cover set) on Y_0.

Case 3.1: $\langle\rangle$: Similarly, the solution is $\{W^* \to a_0 \backsim U_0\}$.

Case 3.2: $b_0 \backsim V_0$: The assumptions are:

$$IsSorted[b_0 \backsim V_0] \wedge b_0 \leq V_0 \wedge IsSorted[V_0] \tag{101}$$

Prove:

$$\mathscr{M}[W^*] = \{\{a_0\}\} \uplus \mathscr{M}[U_0] \uplus \{\{b_0\}\} \uplus \mathscr{M}[V_0] \wedge IsSorted[W^*] \tag{102}$$

Apply **ST-5** (pair multisets) and generate alternatives. Several algorithms are generated. We illustrate a few of them. One version of the algorithm is the efficient classical one:

Algorithm 14. *Merge sorted lists, version 3.*

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

If grouping U_0 with V_0, then the extracted algorithm will be:

Algorithm 15. *Merge sorted lists using insert, version 4.*

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

If grouping U_0, b_0, V_0, then the extracted algorithm will be:

Algorithm 16. *Merge sorted lists using insert, version 5.*

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

Other versions of *Merge*algorithm differ in the last branch of the definition: $Insert[b, Merge[a \frown U, V]]$, $Merge[Insert[a, U], Insert[b, V]]$, etc.

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. 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, 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 using trees, including algorithms which combine operations on several domains.

References

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

[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.

[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.

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

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

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

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

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

[12] I. Dramnesc and T. Jebelean. Automatic Synthesis of Merging and Inserting Algorithms on Lists and Binary Trees using Multisets in *Theorema*. In *SYNASC 2019*, 2019. (submitted).

[13] 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).

[14] I. Dramnesc, T. Jebelean, and S. Stratulat. Theory Exploration of Binary Trees. In *SISY 2015*, pages 139 – 144. IEEE, 2015.

[15] I. Dramnesc, T. Jebelean, and S. Stratulat. Proof-based Synthesis of Sorting Algorithms for Trees. In *LATA 2016*, pages 562–575. Springer, 2016.

[16] 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.

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

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

[19] A. Radoaca. Properties of Multisets Compared to Sets. In *SYNASC 2015*, pages 187–188, 2015.

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