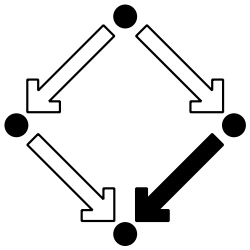
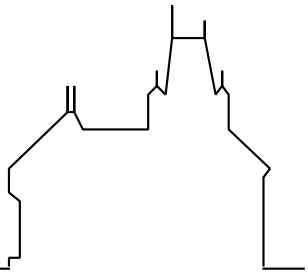


**RISC-Linz**
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe



# Synthesis of Delete on Lists and Binary Trees Using Multisets in Theorema

**Isabela Dramnesc, Tudor Jebelean**

September 2020

RISC-Linz Report Series No. 20-15

# Synthesis of *Delete* on Lists and Binary Trees Using Multisets in *Theorema*

**Isabela Drămnesc**

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

**Tudor Jebelean**

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

*Abstract: We demonstrate the deduction based synthesis of element deletion algorithms for [sorted] lists and [sorted] binary trees, by first developing the necessary theory which is multi–type: basic ordered elements, multisets, lists, and trees. The generated algorithms are "pattern matching", namely sets of conditional equalities, and we also demonstrate their transformation into functional algorithms and, for lists, into tail recursive algorithms. This work constitutes a case study first in theory exploration and second in automated synthesis and transformation of algorithms synthesized on the basis of natural style proofs, which allows to investigate the heuristics of theory construction on multiple types, as well as the natural style inferences and strategies for constructing human readable synthesis proofs. The experiments are performed in the frame of the Theorema system.*

*Keywords: automated reasoning; algorithm synthesis; lists; binary trees; multisets; Theorema*

## 1 Introduction

Finite lists (tuples) and trees (especially search trees) are ubiquitous both in theoretical computer science as well as in numerous applications. Therefore reasoning about algorithms involving them is very important for the safety of computer programs.

Our fist case study on theory exploration and algorithm synthesis involving lists and binary trees using multisets is in [10]. This paper extends [10] by adding the transformations of the synthesized algorithms into functional, tail recursive, and functional tail recursive versions. Also, the proof–based techniques are improved and the generated synthesis proofs are different because the explicit use of the function call instead of metavariables.

The necessary theory is constructed and the proof based synthesis is performed of the algorithms for deletion of the first occurrence and for deletion of all occurrences of an element from a list, a sorted list, a binary tree, and a sorted binary tree.

In order to address in a natural way the notion of *membership* of an element to a composite object and the notion of two composite objects *having the same elements*, *multisets* are used, because these capture appropriately the multiple occurrences of members. Also, in order to create the basis for future research on synthesizing algorithms combining operations on lists and on trees, a multi-sorted theory is developed, containing all the types necessary for such investigations: basic elements from an arbitrary ordered domain (members in composite objects), multisets which characterize the contents of composite objects but abstract their structure, and composite objects (lists and binary trees), which can be sorted.

The authors basic approach to automated synthesis is the one described in their previous work – see e. g. [8, 15]. First a *synthesis conjecture* is produced automatically from the specification (input and output conditions) of the desired algorithm. The conjecture states that for any object satisfying the input condition, there exists an object satisfying the output condition. The proof of the synthesis conjecture is produced automatically, typically using one or more induction principles for the domains of the objects involved, together with certain domain specific inference rules and strategies. From the proof the algorithm is extracted automatically, using the specific witnesses found during the proof process for the existentially quantified goals. The theoretical basis and the correctness of the proof based synthesis scheme is well-known, see [3].

The *algorithms* are produced in a purely logical style: an algorithm consists of a list of clauses, each clause being a (possibly conditional) universally quantified equality which is to be interpreted as a rewrite rule to be applied from left to right. Then, the synthesized algorithms are transformed into their functional, tail recursive, and functional tail recursive versions.

The *practical experiments* are performed in the *Theorema* system [2], which allows to construct and to structure theories in a natural way, to specify natural style provers based on intuitive inference rules and strategies, to present proofs in human readable way, and to execute directly the algorithms described as logical formulae, including the algorithms which are synthesized and their transformations. An illustration of the practical experiments is presented in [**?**].

**Related work.** Multisets play a very important role in science, since any data structure can be represented using multisets. In [17] the authors introduce the axioms, basic notions and some properties for finite multisets, which they call bags. A more extended theory of multisets is described in [1], see also [18] for some interesting

practical developments. For a survey of the literature related to multisets and the use of them see [1]. The theories of lists [6], sets [7] and binary trees [14] are developed in the context of proof–based algorithm synthesis ([4], [5], [8], [15]). The algorithm synthesis method follows the classical proof–based approach as described in [3] and applies explicit induction principles. Our first case study on the synthesis of *Delete* function on lists and binary trees using multisets is in [10]. Later, we use multisets in the entire process of synthesis in order to synthesize sorting algorithms on lists [11], [12], [13], and merging and inserting algorithms on binary trees in [9]. Our first case studies on the transformation of sorting and auxiliary algorithms operating on lists into tail recursive, functional and imperative versions are described in [12].

**The novelty of this paper** in comparison with the authors previous research consists in: the use of multisets in the synthesis problem and in the entire process of algorithm synthesis; the extension of the theories of multisets, lists and binary trees by introducing combined properties which are necessary in the process of algorithm synthesis; the automatically generated proofs of the conjectures in the *Theorema* system; eight case studies on sorted and non–sorted lists and binary trees, and the discovery from proofs of six algorithms which *delete the first occurrence* of an element in a list (and in a binary tree), and *delete all occurrences* respectively; the transformation of the synthesized algorithms into their functional, tail recursive, and functional tail recursive versions for lists (six algorithms are derived: three for *delete the first occurrence* of an element in a list, and three for *delete all occurrences* of an element in a list); and the transformation into functional versions for binary trees (four algorithms are derived: two for *delete the first occurrence* of an element in a binary tree, and two for *delete all occurrences* of an element in a binary tree). This paper extends and improves [10] by adding the transformations of the algorithms following the transformation method in [12], the proof–based techniques are improved and the synthesis proofs are changed by using explicitly the call of the desired function for existential variable.

# 2 Proof–Based Synthesis

This section describes the problem of algorithm synthesis in the context of lists and binary trees and the proof-based synthesis techniques which are used.

## 2.1 Context

### 2.1.1 Notations

According to the *Theorema* style, square brackets have been used for function and for predicate application (e.g., f[x] instead of f(x) and P[a] instead of P(a)). Moreover the quantified variables appear under the quantifier.

In the formulae composing the knowledge there are several kinds of objects: objects from a totally ordered domain which are elements (denoted $a, b, c$) of composite structures, multisets (denoted $A, B, C$), lists/tuples (denoted $U, V, W$), and binary trees (denoted $L, R, S, T$). In general, $X, Y, Z$ are used for all composite objects: multisets, lists, and binary trees. Furthermore meta–variables are starred (e.g., $T^*$, $T_1^*$, $Z^*$) and Skolem constants have integer indices (e.g., $X_0$, $X_1$, $a_0$).

All the statements used at object level in the experiments are formally just predicate logic formulae, and they are called differently depending on their role: *axioms* and *definitions* are used with the obvious meaning; a *property* is a logical consequence of those; and a *conjecture* is something one wants to prove – typically in order to produce and algorithm.

### 2.1.2 Knowledge base

Elements of various composite structures are any objects whose domain is totally ordered, denoted by the usual $\leq$ and $<$. Furthermore multisets, lists, and binary trees are refereed as "composite objects".

The ordering on elements is extended to orderings between an element and a composite object, and between composite objects – see **Definition 3** below.

Concerning the concrete properties related to composite objects, only the ones which are used in the proofs discussed in this paper are listed here.

***Multisets.*** $\emptyset$ is the empty multiset, $\{\{a\}\}$ is the multiset having only the element $a$ with multiplicity 1. The notation for union (additive) [16], difference, subtraction is usual [17], [1], and we also use a "multiple subtraction" operation which deletes all occurrences of an element from a multiset $A \setminus\setminus a$.

**Property 1.** $\underset{A,B,C}{\forall}((A \uplus B) \uplus C = A \uplus (B \uplus C))$

**Property 2.** $\underset{a}{\forall}(\emptyset \setminus \{\{a\}\} = \emptyset)$

**Property 3.** $\underset{a,b,A}{\forall}\left(\begin{array}{l}(\{\{a\}\} \uplus A) \setminus \{\{a\}\} = A \\ (a \neq b) \Longrightarrow (\{\{a\}\} \uplus A) \setminus \{\{b\}\} = \{\{a\}\} \uplus (A \setminus \{\{b\}\})\end{array}\right)$

**Property 4.** $\underset{a,b,A}{\forall}\left(\begin{array}{l}(\{\{a\}\} \uplus A) \setminus\setminus a = A \setminus\setminus a \\ (a \neq b) \Longrightarrow (\{\{a\}\} \uplus A) \setminus\setminus b = \{\{a\}\} \uplus (A \setminus\setminus b)\end{array}\right)$

***Occurrence and multiplicity.*** In order to discuss about occurrence of elements in composite objects one starts from the natural notion of *multiplicity* of an element in a multiset and one extends it to multiplicity of an element in a list, respectively in a binary tree. $M_X[a]$ denotes the multiplicity of an element $a$ in a composite object $X$ (multiset, list, tree). Based on this the notion of equivalence between composite objects is defined as:

**Definition 1.** $\underset{X,Y}{\forall}\Big(\big(X \approx Y\big) \Longleftrightarrow \underset{a}{\forall}\big(M_X[a] = M_Y[a]\big)\Big)$

Obviously this is an equivalence relation when considered over composite objects of the same type, or over an union of types of composite objects as:

Using this basic notion the predicate *member* ($\triangleleft$) for all composite objects is defined.

**Definition 2.** $\underset{a,X}{\forall}\Big(\big(M_X[a] \neq 0\big) \Longleftrightarrow \big(a \triangleleft X\big)\Big)$

Using membership the order on elements is extended to order relations between composite objects[1]:

**Definition 3.** $\underset{a,X,Y}{\forall}\left(\begin{array}{c}\big(a \preceq X\big) \Longleftrightarrow \underset{b \lhd X}{\forall} a \preceq b\big) \\ \big(X \preceq a\big) \Longleftrightarrow \underset{b \lhd X}{\forall} b \preceq a\big) \\ \big(X \preceq Y\big) \Longleftrightarrow \underset{a \lhd X}{\forall}\ \underset{b \lhd Y}{\forall} a \preceq b\big)\end{array}\right)$

***Lists.*** A 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*). List membership can be checked using the properties:

**Property 5.** $\underset{a,b,V}{\forall}\left(\begin{array}{c} a \not\lhd \langle\rangle \\ a \lhd b \smile V \iff (a = b \vee a \lhd V)\end{array}\right)$

**Property 6.** $\underset{a,b,V}{\forall}\big(\big(a \not\lhd b \smile V\big) \Longleftrightarrow \big(a \not\lhd V \wedge a \neq b\big)\big)$

The predicate checking that a list is sorted:

**Definition 4.** $\underset{a,U}{\forall}\left(\begin{array}{c} IsSorted[\langle\rangle] \\ IsSorted[a \smile U] \Longleftrightarrow (a \preceq U \wedge IsSorted[U])\end{array}\right)$

***Trees.*** A tree is either $\varepsilon$ (empty) or a triplet $\langle L,a,R\rangle$, where $L$ and $R$ are trees. Tree membership can be checked using:

**Property 7.** $\underset{a,b,L,R}{\forall}\left(\begin{array}{c} a \not\lhd \varepsilon \\ a \lhd \langle L,b,R\rangle \iff (a = b \vee a \lhd L \vee a \lhd R)\end{array}\right)$

**Property 8.** $\underset{a,b,T,R}{\forall}(a \not\lhd \langle T,b,R\rangle \Longleftrightarrow (a \not\lhd T \wedge a \neq b \wedge a \not\lhd R))$

This predicate checks whether a tree is *sorted* (or *search*, or *ordered*):

**Definition 5.** $\underset{a,L,R}{\forall}\left(\begin{array}{c} IsSorted[\varepsilon] \\ IsSorted[\langle L,a,R\rangle] \Longleftrightarrow IsSorted[L] \wedge IsSorted[R] \wedge L \preceq a \preceq R\end{array}\right)$

**Property 9.** $\underset{a,b,R}{\forall}((a < b \wedge b \preceq R \wedge IsSorted[R]) \Longrightarrow a \not\lhd R))$

**Property 10.** $\underset{a,b,L}{\forall}((a > b \wedge L \preceq b \wedge IsSorted[L]) \Longrightarrow a \not\lhd L))$

Concatenation of trees consists in adding the second tree at the rightmost position of the first:

**Definition 6.** $\underset{a,L,R,S}{\forall}\left(\begin{array}{c} Concat[\varepsilon,R] = R \\ Concat[\langle L,a,R\rangle,S] = \langle L,a,Concat[R,S]\rangle\end{array}\right)$

**Property 11.** $\underset{a,L,R}{\forall}((a \not\lhd L \wedge a \not\lhd R) \Longrightarrow a \not\lhd Concat[L,R])$

[1] This introduces exceptions to transitivity when an empty object is involved.

**Property 12.** $\underset{L,R}{\forall}(IsSorted[L] \uplus IsSorted[R] \Longrightarrow IsSorted[Concat[L,R]])$

***Multisets of lists and trees.*** *The multiset of X* denoted $\mathscr{M}[X]$, is the multiset having the property $X \approx A$.

**Property 13.** $\underset{a,V}{\forall}\begin{pmatrix} \mathscr{M}[\langle\rangle] = \emptyset \\ \mathscr{M}[a \frown V] = \{\{a\}\} \uplus \mathscr{M}[V] \end{pmatrix}$

**Property 14.** $\underset{a,L,R}{\forall}\begin{pmatrix} \mathscr{M}[\varepsilon] = \emptyset \\ \mathscr{M}[\langle L,a,R\rangle] = \mathscr{M}[L] \uplus (\{\{a\}\} \uplus \mathscr{M}[R]) \end{pmatrix}$

***Combined properties.***

**Property 15.**

$$\underset{a,b,X}{\forall}\begin{pmatrix} (a=b) \Longrightarrow (\{\{a\}\} \uplus \mathscr{M}[X]) \setminus \{\{b\}\} = \mathscr{M}[X] \\ (a\neq b) \Longrightarrow (\{\{a\}\} \uplus \mathscr{M}[X]) \setminus \{\{b\}\} = \{\{a\}\} \uplus (\mathscr{M}[X] \setminus \{\{b\}\}) \end{pmatrix}$$

**Property 16.**

$$\underset{a,b,L,R}{\forall}\begin{pmatrix} (a \lhd L) \Longrightarrow \big(\mathscr{M}[L] \uplus \mathscr{M}[R]\big) \setminus \{\{a\}\} = (\mathscr{M}[L] \setminus \{\{a\}\}) \uplus \mathscr{M}[R] \\ (a \lhd R) \Longrightarrow \big(\mathscr{M}[L] \uplus \mathscr{M}[R]\big) \setminus \{\{a\}\} = \mathscr{M}[L] \uplus (\mathscr{M}[R] \setminus \{\{a\}\}) \end{pmatrix}$$

**Property 17.**

$$\underset{\substack{a,b,L,R \\ IsSorted[\langle L,b,R\rangle]}}{\forall}\begin{pmatrix} (a<b) \Longrightarrow \big(\mathscr{M}[\langle L,b,R\rangle] \setminus \{\{a\}\} = (\mathscr{M}[L] \setminus \{\{a\}\}) \uplus (\{\{b\}\} \uplus \mathscr{M}[R])\big) \\ (a=b) \Longrightarrow \big(\mathscr{M}[\langle L,b,R\rangle] \setminus \{\{a\}\} = \mathscr{M}[L] \uplus \mathscr{M}[R]\big) \\ (a>b) \Longrightarrow \big(\mathscr{M}[\langle L,b,R\rangle] \setminus \{\{a\}\} = \mathscr{M}[L] \uplus (\{\{b\}\} \uplus (\mathscr{M}[R] \setminus \{\{a\}\}))\big) \end{pmatrix}$$

**Property 18.** $\underset{L,R}{\forall}(\mathscr{M}[L] \uplus \mathscr{M}[R] = \mathscr{M}[Concat[L,R]])$

**Property 19.** $\underset{a,b,L,T}{\forall}\big((L \preceq b \wedge \mathscr{M}[T] = \mathscr{M}[L] \setminus \{\{a\}\}) \Longrightarrow T \preceq b\big)$

**Property 20.** $\underset{a,b,R,T}{\forall}\big((b \preceq R \wedge \mathscr{M}[T] = \mathscr{M}[R] \setminus \{\{a\}\}) \Longrightarrow b \preceq T\big)$

### 2.1.3 Problem and Approach

*Problem 1:* given an element and a list (or a binary tree), delete the first occurrence of the element in the list (in the binary tree). Moreover, if the original list (tree) is sorted, then the result should be also sorted.

First consider the general case (it is not assumed that the list/tree is sorted). According to the informal specification of the problem, the output condition is $O[a,X,Y]$ : $(\mathscr{M}[Y] = \mathscr{M}[X] \setminus \{\{a\}\})$. The synthesis conjecture states that for any $X$ satisfying the input condition, the result of the function satisfies together with $X$ the output condition:

**Conjecture 1.** $\underset{a,X}{\forall}\Big(\mathscr{M}[F[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\}\Big)$

For sorted objects we have:

**Conjecture 2.** $\underset{a,X}{\forall}\Big(IsSorted[X] \Longrightarrow \mathscr{M}[F[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\} \wedge IsSorted[F[a,X]]\Big)$

*Problem 2:* delete all occurrences of an element from a composite object.

**Conjecture 3.** $\underset{a,X}{\forall}\Big(\mathscr{M}[F[a,X]] = \mathscr{M}[X] \setminus\setminus a\Big)$

**Conjecture 4.** $\underset{a,X}{\forall}\Big(IsSorted[X] \Longrightarrow \mathscr{M}[F[a,X]] = \mathscr{M}[X] \setminus\setminus a \wedge IsSorted[F[a,X]]\Big)$

## 2.2 Induction Principle for Lists

The proof proceeds by setting $a$ arbitrary but fixed, and then the universally quantified formula over lists is proven using the following induction principle for lists:

$$\textbf{Induction-1:}\ \Big(P[\langle\rangle] \wedge \underset{b,U}{\forall}\big(P[U] \Longrightarrow P[b \frown U]\big)\Big) \Longrightarrow \underset{W}{\forall} P[W]$$

One performs induction on variable $U$ from **Conjecture 1** in order to synthesize the *Delete* algorithm as a function $F[a,U]$. (The notations for lists according to the convention is used, in **Conjecture 1** instead of $X$ use $U$ and instead of $Y$ use $V$). The proof is structured as follows:

*Base case:* Prove $O[a,\varepsilon,F[a,\varepsilon]]$. The proof succeeds by reducing the goal to an equation of the form $F[a,\varepsilon] = \Im_1[a]$ (a term possibly depending on $a$).

*Step case:* For arbitrary but fixed $b,U$ assume as induction hypothesis $O[a,U,F[a,U]]$, and prove $O[a,b \frown U,F[a,b \frown U]]$. The proof succeeds by reducing the goal to an equation (possibly prefixed by a condition) of the form $F[a,b \frown U] = \Im_2[a,b,U]$, whose RHS is a term possibly depending on $a,b,U$ and possibly containing $F[a,U]$. The algorithm extracted from the proof has the following structure:

$$\underset{a,b,U}{\forall}\begin{pmatrix} F[a,\langle\rangle] = \Im_1[a] \\ F[a,b \frown U] = \Im_2[a,b,U] \end{pmatrix}$$

In the case of sorted lists (**Conjecture 2**), the proof structure is slightly different:

*Base case:* The proof proceeds in the same way, but one must check additionally that the obtained term represents a sorted list, which is usually trivial.

*Inductive step:* The statement $P[U]$ becomes $IsSorted[U] \Longrightarrow O[a,U,F[a,U]]$. Therefore the induction hypothesis is an implication with LHS $IsSorted[U]$. The goal is an implication with LHS $IsSorted[b \frown U]$, which becomes an additional assumption, and by the definition of *IsSorted* it implies that $U$ is also sorted. By this $O[a,U,F[a,U]]$, becomes a new assumption, thus one arrives exactly in the same situation as in the previous proof, with the only difference that $U$ and $b \frown U$ are assumed to be sorted. Based on this one must prove that the term obtained is also sorted.

## 2.3 Induction Principle for Binary Trees

Similar to the case of lists, one considers the domain specific induction principle for trees in order to prove the universal formula over trees from **Conjecture 1**:

$$\textbf{Induction-2: } \Big(Q[\varepsilon] \wedge \underset{a,L,R}{\forall}\big((Q[L] \wedge Q[R]) \Longrightarrow Q[\langle L,a,R\rangle]\big)\Big) \Longrightarrow \underset{S}{\forall} Q[S]$$

One performs induction on variable $S$ from **Conjecture 1** in order to synthesize the *Delete* algorithm as a function $F[a,S]$. (The notations for trees according to the convention is used, in **Conjecture 1** instead of $X$ use $S$ and instead of $Y$ use $T$). The proof is structured as follows:

*Base case:* For arbitrary but fixed $a$ (new constant), prove $O[a,\varepsilon,F[a,\varepsilon]]$. The proof succeeds by reducing the goal to an equation of the form $F[a,\varepsilon] = \Im_1[a]$ (a term possibly depending on $a$).

*Step case:* For arbitrary but fixed $b$, $L$ and $R$ (new constants), assume: $O[a,L,F[a,T]]$ and $O[a,R,F[a,R]]$ and prove: $O[a,\langle L,b,R\rangle,F[\langle L,b,R\rangle]]$. The proof succeeds by reducing the goal to an equation (possibly prefixed by a condition) of the form $F[a,\langle L,b,R\rangle] = \Im_2[a,b,L,R]$, whose RHS is a term possibly depending on $a,b,L,R$ and possibly containing $F[a,L],F[a,R]$.

The algorithm extracted from the proof has the following structure:

$$\underset{a,b,L,R}{\forall}\begin{pmatrix} F[a,\varepsilon] = \Im_1[a] \\ F[a,\langle L,b,R\rangle] = \Im_2[a,b,L,R] \end{pmatrix}$$

In the case of sorted trees, the situation is the same as in the case of lists: the proof works in the same way, only the given trees $(L,R,F[a,L],F[a,R])$ are assumed to be sorted, and the term obtained has to be also sorted.

## 2.4 Special Inference Rules and Strategies

The strategies are similar to the ones in [8, 15]. The inference rules described in [8, 15]: **IR-1**, **IR-3**, **IR-4**, **IR-5**, **IR-11** are adapted for these current case studies of synthesis and all the others presented in this section are novel.

### 2.4.1 Special Inference Rules

**IR-1:** *Eliminate-assumed-formulae-from-goal*: in a conjunctive goal, eliminate conjuncts which are identical with or instances of assumptions.

**IR-2:** *Rewrite-by-equality.* Example: goal (3) becomes (4).

**IR-3:** *Transform-to-multiple-atoms:* transform parts of the goal or of the assumptions (like e. g. *IsSorted*) into several simpler atoms (e. g. by definition), because then some of the assumptions will match some parts of the conjunctive goal – see **IR-1**. Example: (20) is transformed into (21).

**IR-4:** *Modus Ponens.*

**IR-5:** *Split conjunctive assumptions.*

**IR-6:** *Transform union of MS in goal.* Example: (8) becomes (9).

**IR-7:** *Equality of MS.* Example: if the goal is (5), obtain solution $DelFOL[a_0, b_0 \backsim U_0] = U_0$ and the remaining goal is (6).

**IR-8:** *Reduce the goal using assumptions.* Example: if the goal is (42), using **Property 19**, (35), (33) and the remaining goal will be $a_0 < b_0$.

The following three rules meet strategy **ST-2**.

**IR-9:** *Generate-branches-for-lists.* This rule applies when the proof contains two element constants and generates a branch where they are equal and a branch where they are not. Example: After the goal (23).

**IR-10:** *Generate-branches-for-trees.* Similar to the above.

**IR-10-a**: *for binary sorted trees.* Example: if the goal is: (52) and the local assumptions are: (33), (34), then generate an *AND* node with three branches:
*Branch–1*: the goal is (53), and similarly see *Branch–2*, and *Branch–3*.

**IR-10-b**: *for binary non-sorted trees.* Example: if one has the goal
$\mathscr{M}[DelAOT[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[\langle L_0, b_0, R_0\rangle] \setminus\setminus \{\{a_0\}\}$ and the assumptions are: $\mathscr{M}[DelAOT[a_0, L_0]] = \mathscr{M}[L_0] \setminus\setminus \{\{a_0\}\}$, $\mathscr{M}[DelAOT[a_0, R_0] = \mathscr{M}[R_0] \setminus\setminus \{\{a_0\}\}$, then generate an *AND* node with two branches:
*Branch–1*: the goal becomes
$\mathscr{M}[DelAOT[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[Concat[DelAOT[a_0, L_0], DelAOT[a_0, R_0]]] \wedge$
$a_0 = b_0$,
*Branch–2*: the goal becomes
$\mathscr{M}[DelAOT[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[\langle DelAOT[a_0, L_0], b_0, DelAOT[a_0, R_0]\rangle] \wedge a_0 \neq b_0$.

**IR-11:** *Simple-goal-conditional assumption.* When one has to prove a unique simple ground goal involving only elements and not lists, multisets, binary search trees (e.g., $a < b$, $a > b$, $a = b$, $a \neq b$ which cannot be proved or disproved, then this goal becomes the conditional assumption on the corresponding branch of the synthesized algorithm, and the goal is accepted as proved. *Additionally*, conjuncts including membership of the form $(a_0 \triangleleft L_0 \wedge a_0 \neq b_0)$ and $(a_0 \triangleleft R_0 \wedge a_0 \neq b_0)$ are accepted as conditional assumptions.

**IR-12:** *Eliminate-redundant-ground-goal.* Example: if the goal is $a_0 \neq b_0 \wedge a_0 < b_0$, then is sufficient to prove $a_0 < b_0$.

### 2.4.2 Strategies

**ST-1:** *Quantifier reduction.* This strategy organizes the inference rules for quantifiers (e. g. when applying an induction principle), and it is more effective on goals. Instead of using metavariables for the existential variables in the conjectures, we use the target term expressing the call of the function which is synthesized.

**ST-2:** *Priority-of-Local-Assumptions.* One considers as local assumptions ground formulae which are generated during the current proof and as global assumptions definitions and properties in the knowledge base. The strategy consists in using first the local assumptions. Example: when the goal is (4) and one assumption is (2), then

the new goal becomes (8) because priority is given to terms containing the Skolem constants generated by the induction hypothesis (they correspond to recursive calls).

**ST-3:** *Generate-More-Local-Assumptions.* Example: apply Modus Ponens on local assumptions.

**ST-4:** *Case-Distinction.* When the goal matches a multibranch conditional equality generate several proof branches, follow each branch in turn and produce a set of conditional witnesses which becomes a multiple branch in the synthesized algorithm. The final proof is successful if the disjunction of all conditions is true – this means that the algorithm covers all the possible cases. Example: the goal is (4) and two branches are generated:
*Branch–1* the goal is (5) with condition $a = b$
*Branch–2* the goal is (7) with condition $a \neq b$.

# 3 Experiments on lists

This section describes the discovery of the *Delete* algorithm on lists using multisets.

## 3.1 Synthesis of *Delete* first occurrence

### 3.1.1 Non–sorted lists

As explained in **ST-1** instead of existential variables we use the name of the function which we want to synthesize. **Conjecture 1** becomes:

**Conjecture 5.** $\underset{a,X}{\forall}\Big(\mathscr{M}[DelFOL[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\}\Big)$

The prover automatically generates the proof of **Conjecture 5** by applying **Induction-1** on $X$:

*Proof.* Apply **ST-1** (arbitrary but fixed $a_0$, $b_0$, $U_0$).

*Base case:* The goal is

$$\mathscr{M}[DelFOL[a,\langle\rangle]] = \mathscr{M}[\langle\rangle] \setminus \{\{a_0\}\}. \tag{1}$$

By **Property 2**, **Property 13**, apply **IR-7** and the solution is $DelFOL[a_0,\langle\rangle] = \langle\rangle$.

*Induction step:* Assume:

$$\mathscr{M}[DelFOL[a_0,U_0]] = \mathscr{M}[U_0] \setminus \{\{a_0\}\} \tag{2}$$

and prove:

$$\mathscr{M}[DelFOL[a_0,b_0 \backsim U_0]] = \mathscr{M}[b_0 \backsim U_0] \setminus \{\{a_0\}\}. \tag{3}$$

Apply **IR-2** by **Property 13** and the goal becomes:

$$\mathscr{M}[DelFOL[a_0,b_0 \backsim U_0]] = (\{\{b_0\}\} \uplus \mathscr{M}[U_0]) \setminus \{\{a_0\}\}. \tag{4}$$

Apply **ST-4** by **Property 15** and obtain two branches:

*Branch–1.* Goal (4) becomes:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \mathscr{M}[U_0] \wedge a_0 = b_0. \tag{5}$$

Apply **IR-7**, obtain solution $DelFOL[a_0, b_0 \backsim U_0] = U_0$ and the remaining goal is:

$$a_0 = b_0, \tag{6}$$

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

*Branch–2.* Goal (4) becomes:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = (\{\{b_0\}\} \uplus \mathscr{M}[U_0]) \setminus \{\{a_0\}\} \wedge a_0 \neq b_0. \tag{7}$$

Apply **ST-2**, **IR-2** using the assumption (2) and prove:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \{\{b_0\}\} \uplus \mathscr{M}[DelFOL[a_0, U_0]] \wedge a_0 \neq b_0. \tag{8}$$

Apply **IR-6** by **Property 13** and the goal becomes:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \mathscr{M}[b_0 \backsim DelFOL[a_0, U_0]] \wedge a_0 \neq b_0. \tag{9}$$

Apply **IR-7**, the obtained solutions is $DelFOL[a_0, b_0 \backsim U_0] = b_0 \backsim DelFOL[a_0, U_0]$ and the remaining goal is:

$$a_0 \neq b_0, \tag{10}$$

which according to **IR-11** becomes the conditional assumption on this branch. □

From the proof the following algorithm is extracted:

**Algorithm 1.**

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

**The functional version**

In order to obtain the functional program one uses the functions which are reverse to the pattern $b \backsim U$ used in the algorithms, namely *head* and *Tail*.

**Algorithm 2.**

$$\underset{a,U}{\forall} \left( DelFOLf[a, U] = \left\{ \begin{array}{ll} \langle\rangle, & \text{if } U = \langle\rangle \\ Tail[U], & \text{if } a = head[U] \\ head[U] \backsim DelFOLf[a, Tail[U]], & \text{if } a \neq head[U] \end{array} \right. \right)$$

**The tail recursive version**

The algorithm is transformed into its tail recursive version by adding a third argument as "accumulator" for the result. In order to construct this accumulator, one needs to reverse the operation of the original algorithm: if in *DelFOL* the result is constructed by $\frown$ (cons), in the tail recursive version it must be constructed by the dual operation $\backsim$ (add to the end) which places an element at the end of a list and the operation $\asymp$ (concat) which concatenates two lists. These functions are implemented in *Theorema* in an efficient manner.

**Algorithm 3.**

$$\underset{a,b,U,V}{\forall}\left(\begin{array}{c} DelFOL[a,U] = DelFOLtr[U,a,\langle\rangle] \\ DelFOLtr[\langle\rangle,a,V] = V \\ DelFOLtr[b \frown U,a,V] = \left\{\begin{array}{ll} V \asymp U, & \text{if } a = b \\ DelFOLtr[U,a,V \backsim b], & \text{if } a \neq b \end{array}\right. \end{array}\right)$$

**The functional tail recursive version**

**Algorithm 3** is transformed into:

**Algorithm 4.**

$$\underset{a,U,V}{\forall}\left(\begin{array}{l} DelFOL[a,U] = DelFOLtrf[U,a,\langle\rangle] \\ DelFOLtrf[U,a,V] = \\ \left\{\begin{array}{ll} V & \text{if } U = \langle\rangle \\ V \asymp Tail[U], & \text{if } a = head[U] \\ DelFOLtrf[Tail[U],a,V \backsim head[U]], & \text{if } a \neq head[U] \end{array}\right. \end{array}\right)$$

### 3.1.2 Sorted lists

If the input list is assumed to be sorted, one has to prove **Conjecture 2** which becomes:

**Conjecture 6.**

$$\underset{a,X}{\forall} IsSorted[X] \Longrightarrow \Big(\mathscr{M}[DelFOL[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\} \wedge IsSorted[DelFOL[a,X]]\Big)$$

The proof is similar with the previous proof, applies **Induction-1** and in addition one uses **IR-8**.

*Proof. Base case:* The obtained solution is $DelFOL[a_0,\langle\rangle] = \langle\rangle$.

*Induction step:* Apply **IR-4**, **IR-5**, **IR-3** and the assumptions are:

$$\mathscr{M}[DelFOL[a_0,U_0]] = \mathscr{M}[U_0] \setminus \{\{a_0\}\}, \tag{11}$$

$$IsSorted[DelFOL[a_0,U_0]] \wedge b_0 \preceq U_0 \wedge IsSorted[U_0]. \tag{12}$$

The goal is:

$$\begin{array}{l} \mathscr{M}[DelFOL[a_0,b_0 \frown U_0]] = \mathscr{M}[b_0 \frown U_0] \setminus \{\{a_0\}\} \wedge \\ \qquad\qquad IsSorted[DelFOL[a_0,b_0 \frown U_0]]. \end{array} \tag{13}$$

Apply **IR-2** by **Property 13** and the goal becomes:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = (\{\{b_0\}\} \uplus \mathscr{M}[U_0]) \setminus \{\{a_0\}\} \wedge \\ IsSorted[DelFOL[a_0, b_0 \backsim U_0]]. \tag{14}$$

Apply **ST-4** by **Property 15** and generate two branches:

*Branch–1.* The goal is

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \mathscr{M}[U_0] \wedge IsSorted[DelFOL[a_0, b_0 \backsim U_0]] \wedge a_0 = b_0 \tag{15}$$

Apply **IR-7**, the obtained solution is $DelFOL[a_0, b_0 \backsim U_0] = U_0$ and the new goal is:

$$IsSorted[U_0] \wedge a_0 = b_0. \tag{16}$$

Apply **IR-1** using (12) and the proof succeeds on this branch with the conditional assumption $a_0 = b_0$ (**IR-11**).

*Branch–2.* The goal is

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \{\{b_0\}\} \uplus (\mathscr{M}[U_0] \setminus \{\{a_0\}\}) \wedge \\ IsSorted[DelFOL[a_0, b_0 \backsim U_0]] \wedge a_0 \neq b_0. \tag{17}$$

Apply **IR-2** using (11) and the goal is:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \{\{b_0\}\} \uplus \mathscr{M}[DelFOL[a_0, U_0]] \wedge \\ IsSorted[DelFOL[a_0, b_0 \backsim U_0]] \wedge a_0 \neq b_0. \tag{18}$$

Apply **IR-6** by **Property 13** and the goal becomes:

$$\mathscr{M}[DelFOL[a_0, b_0 \backsim U_0]] = \mathscr{M}[b_0 \backsim DelFOL[a_0, U_0]] \wedge \\ IsSorted[DelFOL[a_0, b_0 \backsim U_0]] \wedge a_0 \neq b_0. \tag{19}$$

Apply **IR-7**, the obtained solution is $DelFOL[a_0, b_0 \backsim U_0] = b_0 \backsim DelFOL[a_0, U_0]$ and the new goal is:

$$IsSorted[b_0 \backsim DelFOL[a_0, U_0]] \wedge a_0 \neq b_0. \tag{20}$$

Apply **IR-3** and the goal becomes:

$$b_0 \preceq DelFOL[a_0, U_0] \wedge IsSorted[DelFOL[a_0, U_0]] \wedge a_0 \neq b_0. \tag{21}$$

Apply **IR-8** with the assumptions (11), **IR-1** using (12) and the proof succeeds with the conditional assumption $a_0 \neq b_0$ (**IR-11**). □

From this proof the same **Algorithm 1** is extracted.

## 3.2 Synthesis of *Delete* all occurrences

### 3.2.1 Non–sorted lists

**Conjecture 3** becomes:

**Conjecture 7.** $\underset{a,X}{\forall}\Big(\mathscr{M}[DelAOL[a,X]] = \mathscr{M}[X] \setminus\setminus a\Big)$

Prove **Conjecture 7** by applying **Induction-1** on $X$. The proof proceeds similarly as for delete first occurrence, in addition one uses **Property 6** and **IR-9**.

*Proof. Base case:* The solution is $DelAOL[a_0, \langle\rangle] = \langle\rangle$.

*Induction step:* Apply **IR-4**, **IR-5**, **IR-3**. The assumption is:

$$\mathscr{M}[DelAOL[a_0, U_0]] = \mathscr{M}[U_0] \setminus\setminus a_0 \tag{22}$$

and the goal is:

$$\mathscr{M}[DelAOL[a_0, b_0 \frown U_0]] = \mathscr{M}[b_0 \frown U_0] \setminus\setminus a_0. \tag{23}$$

Apply **IR-9** and two branches are generated:

*Branch–1.* The goal (23) becomes:

$$\mathscr{M}[DelAOL[b_0, a_0 \frown U_0]] = (\{\{b_0\}\} \uplus \mathscr{M}[U_0]) \setminus\setminus a_0 \ \wedge\ a_0 = b_0. \tag{24}$$

By **Property 4** (first part) through equality rewriting the goal changes to:

$$\mathscr{M}[DelAOL[a_0, b_0 \frown U_0]] = \mathscr{M}[U_0] \setminus\setminus a_0 \ \wedge\ a_0 = b_0 \tag{25}$$

and by rewriting again using the induction hypothesis:

$$\mathscr{M}[DelAOL[b_0, a_0 \frown U_0]] = \mathscr{M}[DelAOL[a_0, U_0]] \ \wedge\ a_0 = b_0. \tag{26}$$

Apply **IR-7**, the solution is $DelAOL[a_0, b_0 \frown U_0] = DelAOL[a_0, U_0]$ and the goal is:

$$a_0 = b_0. \tag{27}$$

Apply **IR-11** and the proof succeeds on this branch with the conditional assumption $a_0 = b_0$.

*Branch–2.* The goal is

$$\mathscr{M}[DelAOL[b_0, a_0 \frown U_0]] = (\{\{b_0\}\} \uplus \mathscr{M}[U_0]) \setminus\setminus a_0 \ \wedge\ a_0 \neq b_0. \tag{28}$$

By **Property 4** (second part) through equality rewriting te goal changes to:

$$\mathscr{M}[DelAOL[b_0, a_0 \frown U_0]] = \{\{b_0\}\} \uplus (\mathscr{M}[U_0] \setminus\setminus a_0) \ \wedge\ a_0 \neq b_0 \tag{29}$$

and by rewriting again using the induction hypothesis:

$$\mathscr{M}[DelAOL[b_0, a_0 \frown U_0]] = \{\{b_0\}\} \uplus \mathscr{M}[DelAOL[a_0, U_0]] \ \wedge\ a_0 \neq b_0. \tag{30}$$

By grouping multiset terms we obtain:

$$\mathscr{M}[DelAOL[b_0, a_0 \backsim U_0]] = \mathscr{M}[b_0 \backsim DelAOL[a_0, U_0]] \ \wedge \ a_0 \neq b_0. \tag{31}$$

Apply **IR-7**, the solution is $DelAOL[a_0, b_0 \backsim U_0] = b_0 \backsim DelAOL[a_0, U_0]$ and the new goal is:

$$a_0 \neq b_0. \tag{32}$$

Apply **IR-11** and the proof succeeds on this branch with the conditional assumption $a_0 \neq b_0$. □

The following algorithm is extracted from the proof:

**Algorithm 5.**

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

**The functional version**

**Algorithm 6.**

$$\underset{a,U}{\forall} \left( DelAOLf[a, U] = \left\{ \begin{array}{ll} \langle\rangle, & \text{if } U = \langle\rangle, \\ DelAOLf[a, Tail[U]], & \text{if } a = head[U] \\ head[U] \backsim DelAOLf[a, Tail[U]], & \text{if } a \neq head[U] \end{array} \right. \right)$$

**The tail recursive version**

**Algorithm 7.**

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

**The functional tail recursive version**

**Algorithm 8.**

$$\underset{a,b,U,V}{\forall} \left( \begin{array}{c} DelAOL[a, U] = DelAOLtrf[U, a, \langle\rangle] \\ DelAOLtrf[U, a, V] = \left\{ \begin{array}{ll} V, & \text{if } U = \langle\rangle \\ DelAOLtrf[Tail[U], a, V], & \text{if } a = head[U] \\ DelAOLtrf[Tail[U], a, V \frown b], & \text{if } a \neq head[U] \end{array} \right. \end{array} \right)$$

### 3.2.2 Sorted lists

**Conjecture 4** becomes:

**Conjecture 8.** $\underset{a,X}{\forall} IsSorted[X] \Longrightarrow$

$$\Big( \mathscr{M}[DelAOL[a, X]] = \mathscr{M}[X] \setminus\setminus \{\{a\}\} \wedge IsSorted[DelAOL[a, X]] \Big)$$

The proof starts by applying **Induction-1** on $X$ and is similar with the previous ones. From the proof the same **Algorithm 5** is extracted.

# 4 Experiments on binary trees

This subsection describes the discovery of the *Delete* algorithm on binary trees using multisets.

## 4.1 Synthesis of *Delete* first occurrence

### 4.1.1 Binary sorted tree

**Conjecture 2** becomes:

**Conjecture 9.** $\underset{a,X}{\forall} IsSorted[X] \Longrightarrow$
$\Big(\mathscr{M}[DelFOST[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\} \wedge IsSorted[DelFOST[a,X]]\Big)$

The prover generates the proof of **Conjecture 9** by applying **Induction-2** on $X$.

*Proof. Base case:* By **Property 14**, **Property 2** the solution is $DelFOST[a,\varepsilon] = \varepsilon$.

*Induction step:* Apply **IR-4, IR-5, IR-3** and the assumptions are:

$$\mathscr{M}[DelFOST[a_0,L_0]] = \mathscr{M}[L_0] \setminus \{\{a_0\}\} \ \wedge \ IsSorted[DelFOST[a_0,L_0]], \tag{33}$$

$$\mathscr{M}[DelFOST[a_0,R_0]] = \mathscr{M}[R_0] \setminus \{\{a_0\}\} \ \wedge \ IsSorted[DelFOST[a_0,R_0]], \tag{34}$$

$$IsSorted[\langle L_0,b_0,R_0\rangle] \ \wedge \ IsSorted[L_0] \ \wedge \ L_0 \preceq b_0 \ \wedge \ b_0 \preceq R_0 \ \wedge \ IsSorted[R_0], \tag{35}$$

and the goal is:

$$\begin{aligned}\mathscr{M}[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] = \mathscr{M}[\langle L_0,b_0,R_0\rangle] \setminus \{\{a_0\}\}) \wedge \\ IsSorted[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]].\end{aligned} \tag{36}$$

Apply **ST-4** by **Property 17** and generate three branches:

*Branch–1.* Goal (36) becomes:

$$\begin{aligned}&\mathscr{M}[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] = (\mathscr{M}[L_0] \setminus \{\{a_0\}\}) \uplus (\{\{b_0\}\} \uplus \mathscr{M}[R_0]) \wedge \\ &IsSorted[\langle L_0,b_0,R_0\rangle] \ \wedge \ IsSorted[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] \ \wedge \ a_0 < b_0.\end{aligned} \tag{37}$$

Apply **IR-1** using (35) and **IR-2** using (33) and the goal becomes:

$$\begin{aligned}\mathscr{M}[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] = \mathscr{M}[DelFOST[a_0,L_0]] \uplus (\{\{b_0\}\} \uplus \mathscr{M}[R_0]) \wedge \\ IsSorted[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] \ \wedge \ a_0 < b_0.\end{aligned} \tag{38}$$

Apply **IR-6** and the new goal is:

$$\begin{aligned}\mathscr{M}[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] = \mathscr{M}[\langle DelFOST[a_0,L_0],b_0,R_0\rangle] \wedge \\ IsSorted[DelFOST[a_0,\langle L_0,b_0,R_0\rangle]] \ \wedge \ a_0 < b_0.\end{aligned} \tag{39}$$

Apply **IR-7**, the solution is $DelFOST[a_0, \langle L_0, b_0, R_0\rangle] = \langle DelFOST[a_0, L_0], b_0, R_0\rangle$ and the new goal is:

$$IsSorted[\langle DelFOST[a_0, L_0], b_0, R_0\rangle] \wedge a_0 < b_0. \tag{40}$$

Apply **IR-3** and the goal becomes:

$$IsSorted[DelFOST[a_0, L_0]] \wedge DelFOST[a_0, L_0] \preceq b_0 \wedge b_0 \preceq R_0 \wedge IsSorted[R_0] \wedge a_0 < b_0. \tag{41}$$

Apply **IR-1** using (33) and (35) and the new goal is:

$$DelFOST[a_0, L_0] \preceq b_0 \wedge a_0 < b_0. \tag{42}$$

Apply **IR-8** using **Property 19**, (35), (33) and the proof succeeds on this branch with the conditional assumption $a_0 < b_0$ (**IR-11**).

*Branch–2.* Goal (36) becomes:

$$\mathscr{M}[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[L_0] \uplus \mathscr{M}[R_0] \wedge IsSorted[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] \wedge IsSorted[\langle L_0, b_0, R_0\rangle] \wedge a_0 = b_0. \tag{43}$$

Apply **IR-1** using (35) and by **Property 18** the new goal is:

$$\mathscr{M}[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[Concat[L_0, R_0]] \wedge IsSorted[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] \wedge a_0 = b_0. \tag{44}$$

Apply **IR-7**, obtain the solution $DelFOST[a_0, \langle L_0, b_0, R_0\rangle] = Concat[L_0, R_0]$ and the new goal is:

$$IsSorted[Concat[L_0, R_0]] \wedge a_0 = b_0. \tag{45}$$

Apply **IR-2** by **Property 12**, **IR-1** using (35) and the proof succeeds on this branch with the conditional assumption $a_0 = b_0$ (**IR-11**).

*Branch–3.* Goal (36) becomes:

$$\mathscr{M}[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[L_0] \uplus (\{\{b_0\}\} \uplus (\mathscr{M}[R_0] \setminus \{\{a_0\}\})) \wedge IsSorted[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] \wedge IsSorted[\langle L_0, b_0, R_0\rangle] \wedge a_0 > b_0. \tag{46}$$

Apply **IR-2** using (34) and **IR-1** using (35) and the goal becomes:

$$\mathscr{M}[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[L_0] \uplus (\{\{b_0\}\} \uplus \mathscr{M}[DelFOST[a_0, R_0]]) \wedge IsSorted[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] \wedge a_0 > b_0. \tag{47}$$

Apply **IR-6** and the new goal is:

$$\mathscr{M}[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[\langle L_0, b_0, DelFOST[a_0, R_0]\rangle] \wedge IsSorted[DelFOST[a_0, \langle L_0, b_0, R_0\rangle]] \wedge a_0 > b_0. \tag{48}$$

Similar as before apply **IR-7**, the obtained solution is $DelFOST[a_0, \langle L_0, b_0, R_0\rangle] = \langle L_0, b_0, DelFOST[a_0, R_0]\rangle$, apply **IR-3**, **IR-1** using (35) and (34) and the goal is:

$$b_0 \preceq DelFOST[a_0, R_0] \ \wedge \ a_0 > b_0. \tag{49}$$

Apply **IR-8** using **Property 20**, (35), (34) and the proof succeeds on this branch with the conditional assumption $a_0 > b_0$ (**IR-11**). □

The following algorithm is extracted from the proof:

**Algorithm 9.**

$$\underset{a,b,L,R}{\forall}\left(\begin{array}{c} DelFOST[a,\varepsilon] = \varepsilon \\ DelFOST[a,\langle L,b,R\rangle] = \left\{\begin{array}{ll} \langle DelFOST[a,L],b,R\rangle, & \text{if } a<b \\ Concat[L,R], & \text{if } a=b \\ \langle L,b,DelFOST[a,R]\rangle, & \text{if } a>b \end{array}\right. \end{array}\right)$$

**The functional version**

**Algorithm 10.**

$$\underset{a,T}{\forall}\left(\begin{array}{l} DelFOSTf[a,T] = \\ \left\{\begin{array}{ll} \varepsilon, & \text{if } T=\varepsilon \\ \langle DelFOSTf[a,Left[T]], root[T], Right[T]\rangle, & \text{if } a<root[T] \\ Concat[Left[T], Right[T]], & \text{if } a=root[T] \\ \langle Left[T], root[T], DelFOSTf[a,Right[T]]\rangle, & \text{if } a>root[T] \end{array}\right. \end{array}\right)$$

where *Left*, *Right* are functions which return the left subtree, and the right subtree of a tree, and *root* is the function which returns the root of a tree.

### 4.1.2 Binary tree not sorted

**Conjecture 1** becomes:

**Conjecture 10.** $\underset{a,X}{\forall}\Big(\mathscr{M}[DelFOT[a,X]] = \mathscr{M}[X] \setminus \{\{a\}\}\Big)$

The proof is similar with the previous one by applying **Induction-2** on $X$, in addition one uses **Property 16**. The extracted algorithm from the proof is:

**Algorithm 11.**

$$\underset{a,b,L,R}{\forall}\left(\begin{array}{c} DelFOT[a,\varepsilon] = \varepsilon \\ DelFOT[a,\langle L,b,R\rangle] = \left\{\begin{array}{ll} Concat[L,R], & \text{if } a=b \\ \langle L,b,DelFOT[a,R]\rangle, & \text{if } a\neq b \wedge a \lhd R \\ \langle DelFOT[a,L],b,R\rangle, & \text{if } a\neq b \wedge a \lhd L \\ \langle L,b,R\rangle, & \text{otherwise} \end{array}\right. \end{array}\right)$$

This algorithm is not efficient, but this was expected since the trees are not sorted.

**The functional version**

**Algorithm 12.**

$$\underset{a,T}{\forall}\left(\begin{array}{l} DelFOTf[a,T] = \\ \left\{\begin{array}{ll} \varepsilon, & \text{if } T = \varepsilon \\ Concat[Left[T], Right[T]], & \text{if } a = root[T] \\ \langle Left[T], root[T], DelFOTf[a, Right[T]]\rangle, & \text{if } a \neq root[T] \wedge a \triangleleft Right[T] \\ \langle DelFOTf[a, Left[T]], root[T], Right[T]\rangle, & \text{if } a \neq root[T] \wedge a \triangleleft Left[T] \\ T, & \text{otherwise} \end{array}\right. \end{array}\right)$$

## 4.2 Synthesis of Delete all occurrences

### 4.2.1 Sorted tree

**Conjecture 4** becomes:

**Conjecture 11.** $\underset{a,X}{\forall} IsSorted[X] \Longrightarrow$

$$\Big(\mathscr{M}[DelAOST[a,X]] = \mathscr{M}[X] \setminus\setminus \{\{a\}\} \ \wedge \ IsSorted[DelAOST[a,X]]\Big)$$

The proof starts by applying **Induction-2** and is similar with the previous ones. In addition one uses **IR-10-a**, **Property 8**, **Property 9**, **Property 11**, and **Property 10**.

*Proof. Base case:* The solution is $DelAOST[a,\varepsilon] = \varepsilon$.

*Induction step:* Apply **IR-4, IR-5, IR-3** and the assumptions are similar to (33), (34), but using multiple subtraction ($A \setminus\setminus a$) instead of element subtraction ($A \setminus \{\{a\}\}$), namely:

$$\mathscr{M}[DelAOST[a_0, L_0]] = \mathscr{M}[L_0] \setminus\setminus \{\{a_0\}\} \ \wedge \ IsSorted[DelAOST[a_0, L_0]], \tag{50}$$

$$\mathscr{M}[DelAOST[a_0, R_0]] = \mathscr{M}[R_0] \setminus\setminus \{\{a_0\}\} \ \wedge \ IsSorted[DelAOST[a_0, R_0]], \tag{51}$$

and (35). The goal is:

$$\begin{aligned} \mathscr{M}[DelAOST[a_0, \langle L_0, b_0, R_0\rangle]] = \mathscr{M}[\langle L_0, b_0, R_0\rangle] \setminus\setminus a_0 \wedge \\ IsSorted[DelAOST[a_0, \langle L_0, b_0, R_0\rangle]]. \end{aligned} \tag{52}$$

Apply **IR-10-a** using the local assumptions above and obtain three branches:

*Branch–1.* Apply **IR-7**, obtain $DelAOST[a_0, \langle L_0, b_0, R_0\rangle] = \langle DelAOST[a_0, L_0], b_0, R_0\rangle$ as solution and prove:

$$IsSorted[\langle DelAOST[a_0, L_0], b_0, R_0\rangle] \ \wedge \ a_0 < b_0. \tag{53}$$

By **IR-3** using the assumptions (35), (50) and **IR-12** the new goal is:

$$DelAOST[a_0, L_0] \preceq b_0 \ \wedge \ a_0 < b_0. \tag{54}$$

Apply **IR-8** by **Property 9** using (35), (50), and the goal becomes:

$$a_0 < b_0, \tag{55}$$

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

*Branch–2.* Obtain the solution $DelAOST[a_0, \langle L_0, b_0, R_0\rangle] = Concat[DelAOST[a_0, L_0], DelAOST[a_0, R_0]]$ and similarly, by **Property 11**, **Property 12** the proof succeeds on this branch with the conditional assumption $a_0 = b_0$ (**IR-11**).

*Branch–3.* Obtain the solution $DelAOST[a_0, \langle L_0, b_0, R_0\rangle] = \langle L_0, b_0, DelAOST[a_0, R_0]\rangle$. Similarly, by **Property 10**, **Property 8** the proof succeeds on this branch with the conditional assumption $a_0 > b_0$ (**IR-11**). □

The extracted algorithm from the proof is:

**Algorithm 13.**

$$\underset{a,b,L,R}{\forall}\left(\begin{array}{l} DelAOST[a,\varepsilon] = \varepsilon \\ DelAOST[a,\langle L,b,R\rangle] = \\ \left\{\begin{array}{ll} \langle DelAOST[a,L],b,R\rangle, & \text{if } a<b \\ Concat[DelAOST[a,L],DelAOST[a,R]], & \text{if } a=b \\ \langle L,b,DelAOST[a,R]\rangle, & \text{if } a>b \end{array}\right. \end{array}\right)$$

**The functional version**

**Algorithm 14.**

$$\underset{a,T}{\forall}\left(\begin{array}{l} DelAOSTf[a,T] = \\ \left\{\begin{array}{ll} \varepsilon, & \text{if } T=\varepsilon \\ \langle DelAOSTf[a,Left[T]],root[T],Right[T]\rangle, & \text{if } a<root[T] \\ Concat[DelAOSTf[a,Left[T]],DelAOSTf[a,Right[T]]], & \text{if } a=root[T] \\ \langle Left[T],root[T],DelAOSTf[a,Right[T]]\rangle, & \text{if } a>root[T] \end{array}\right. \end{array}\right)$$

### 4.2.2 Nonsorted tree

**Conjecture 3** becomes:

**Conjecture 12.** $\underset{a,X}{\forall}\Big(\mathscr{M}[DelAOT[a,X]] = \mathscr{M}[X] \setminus\setminus a\Big)$

The proof starts by applying **Induction-2** on $X$ and is similar with the previous one. In addition one uses **IR-10-b**. The extracted algorithm is:

**Algorithm 15.**

$$\underset{a,b,L,R}{\forall}\left(\begin{array}{c} DelAOT[a,\varepsilon] = \varepsilon \\ DelAOT[a,\langle L,b,R\rangle] = \\ \left\{\begin{array}{ll} Concat[DelAOT[a,L],DelAOT[a,R]], & \text{if } a=b \\ \langle DelAOT[a,L],b,DelAOT[a,R]\rangle, & \text{if } a\neq b \end{array}\right. \end{array}\right)$$

### The functional version

**Algorithm 16.**

$$\underset{a,T}{\forall}\left(\begin{array}{l} DelAOTf[a,T] = \\ \left\{\begin{array}{ll} \varepsilon, & \text{if } T = \varepsilon \\ Concat[DelAOTf[a,Left[T]],DelAOTf[a,Right[T]]], & \text{if } a = root[T] \\ \langle DelAOTf[a,Left[T]],root[T],DelAOTf[a,Right[T]]\rangle, & \text{if } a \neq root[T] \end{array}\right. \end{array}\right)$$

### Conclusions

The experiments presented in this paper demonstrate how the construction of the appropriate theory for lists and binary trees containing elements from an ordered domain can be performed in an intuitive and natural way by using a multi-type approach involving multisets.

We also demonstrate how this theory can be used as the background for proof-based synthesis of algorithms for element deletion. While the algorithms themselves are rather straightforward, the process of proving is still very interesting, because it reveals important heuristics (in form of inference rules and strategies) for constructing the proofs in natural style. In this paper we describe 16 algorithms for element deletion: 8 algorithms operating on sorted and non–sorted lists, and 8 algorithms operating on sorted and non–sorted binary trees. From the 8 proofs generated automatically in *Theorema* 6 algorithms are discovered from proofs, each one is based on the proof of a specific conjecture: 2 on lists, and 4 on binary trees. 10 algorithms are obtained by transforming the synthesized ones into their functional, tail recursive, and functional tail recursive versions: 6 algorithms on lists, and 4 algorithm (the functional versions) on binary trees.

The experiments also demonstrate how these algorithms work in practice: the algorithms are produced in form of logical formulae, which can be executed (applied to concrete arguments) in the *Theorema* system. (In fact translation from this form into imperative style is straightforward.)

The case study on the synthesis of the algorithms presented here constitutes the first step towards the synthesis of more complex algorithms, in particular for sorting of lists and binary trees.

In contrast to our previous research, where this investigation was performed only using the domain of binary trees, in this new approach we plan to study more sophisticated algorithms which also transform trees into lists and back, and to extend the automation approach to the characterization of the efficiency of algorithms.

### References

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

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

[3] A. Bundy, L. Dixon, J. Gow, and J. Fleuriot. Constructing induction rules for deductive synthesis proofs. *ENTCS*, 153:3–21, March 2006.

[4] I. Dramnesc and T. Jebelean. Proof techniques for synthesis of sorting algorithms. In *SYNASC 2011*, pages 101–109. IEEE Computer Society, 2011.

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

[6] I. Dramnesc and T. Jebelean. Theory exploration in Theorema: Case study on lists. In *SACI 2012*, pages 421 – 426. IEEE Xplore, 2012.

[7] I. Dramnesc and T. Jebelean. Theory exploration of sets represented as monotone lists. In *SISY 2014*, pages 163 – 168. IEEE Xplore, 2014.

[8] I. Dramnesc and T. Jebelean. Synthesis of list algorithms by mechanical proving. *Journal of Symbolic Computation*, 68:61–92, 2015.

[9] I. Dramnesc and T. Jebelean. Automatic synthesis of merging and inserting algorithms on binary trees using multisets in *theorema*. In *MACIS 2019*, LNCS, pages 153–168. Springer, 2019.

[10] 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*, pages 213–220. IEEE Xplore, 2019.

[11] I. Dramnesc and T. Jebelean. Proof–based synthesis of sorting algorithms using multisets in *Theorema*. In *FROM 2019*, pages 76–91. EPTCS 303, 2019.

[12] I. Dramnesc and T. Jebelean. Deductive synthesis of Bubble-Sort using multisets. In *SAMI 2020*, pages 165–172. IEEE, 2020.

[13] I. Dramnesc and T. Jebelean. Deductive synthesis of Min-Max-Sort using multisets. In *SACI 2020*, pages 165–172. IEEE, 2020.

[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. Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques. *Journal of Symbolic Computation*, 90:3–41, 2019.

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

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

[18] A. Radoaca. Properties of multisets compared to sets. In *SYNASC 2015*, pages 187–188. IEEE Computer Society, 2015.