A Case Study on Algorithm Discovery from Proofs: The *Insert* Function on Binary Trees

Isabela Drămnesc
Department of Computer Science
West University
Timişoara, Romania
Email: idramnesc@info.uvt.ro

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

Sorin Stratulat
LITA, Department of Computer Science
Université de Lorraine
Metz, France
Email: sorin.stratulat@univ-lorraine.fr

***Abstract*—We present a proof–based automatic synthesis experiment in the context of sorting binary trees, namely the synthesis of the function which inserts an element in a sorted binary tree at the appropriate place. The algorithm is automatically extracted from the automatically produced proof of the conjecture which expresses the existence of the desired output for each appropriate input of the function.**

This is a case study with the purpose of finding and illustrating general and domain specific inference rules and strategies for efficiently producing proofs from which the desired algorithms can be extracted.

The construction of the necessary theory, of the provers, and the experiment itself are performed in the *Theorema* system.

***Keywords*—automated reasoning, binary trees, Theorema**

I. Introduction

Certified software is usually based on formal verification: by using an appropriate verification condition generator, a set of logical statements is produced from the already existing software (or algorithm) and then these are proved using automated theorem proving or checked using model checking.

An interesting alternative approach is to automatically synthesize the algorithm (and then the software) using a formal environment. We illustrate in this paper the proof–based algorithm synthesis. Namely, one starts from the properties of the desired function, and then proves automatically the conjecture: "for each appropriate input there exists an output with the desired properties". The algorithm is extracted from the proof as a set of conditional equalities. Proof–based synthesis is a classical method (see e.g. [6]), however concrete experiments with non–trivial algorithms are scarce.

We present in this paper a case study on the automatic generation of the *Insert* function for the sorting of binary trees. In the insert–sort algorithm, this function inserts an element in a sorted binary tree, such that the resulting tree is also sorted.

The purpose of the case study is to discover and illustrate the proving methods which are successful in producing efficiently a proof from which the algorithm can be extracted. Furthermore the experiment illustrates the process of constructing the theory which is necessary for expressing the specification of the desired function and the properties of the objects involved – which are needed for the success of the proofs.

The construction of the theory, of the appropriate prover, and the proof experiments are performed in the *Theorema* system [3], which is implemented on top of *Mathematica* [17]. The *Theorema* system allows to express and construct the logical formulae and the proofs in natural style (that is, similar to the style humans use), and furthermore the automatically synthesized algorithms are directly executable in the system.

A. Related Work

In literature there are several methods for recursive program synthesis. Three of them: constructive/deductive synthesis, schema-based synthesis and inductive synthesis have been compared in [1] in the synthesis of a common program. A survey of logic program synthesis has been done in [7] and [14]. In the context of constructive/deductive synthesis, the approach in this paper is proof–based and describes deductive techniques for synthesizing the *Insert* algorithm operating on binary trees.

In [5] the authors describe some techniques (based on a combination of "rippling" [4] and "middle–out reasoning" [16]) to construct induction rules for deductive synthesis. In contrast, in this paper we give the induction principle and we apply some special strategies and techniques in order to obtain the algorithm.

Most of the techniques from literature have been applied for synthesizing algorithms operating on natural numbers or lists, but none on binary trees.

In classical approaches (see e. g. [15]) problems like sorting of trees and inserting an element in a sorted tree are not investigated.

In this paper we handle the problem of inserting an element in a sorted tree such that the result is sorted. From the specification of the problem, we derive a conjecture which is automatically proved by our new prover implemented in the *Theorema* system. The prover uses techniques which are specific to the domain of binary trees. Some of these techniques were introduced in [9] and [13] and some of them are introduced in this paper. From the generated proof, the corresponding algorithm is automatically extracted. The corresponding theory of binary trees is explored in [12].

Compared to our previous work on algorithm synthesis on lists [8], in this paper we use a similar proof–based approach, but different techniques to synthesize algorithms on binary

trees. The authors apply in [9] combinatorial techniques in order to discover from proofs auxiliary algorithms, other than *Insert*, which are versions of an algorithm that merges two sorted binary trees into a sorted one. All these auxiliary algorithms are necessary in the sorting algorithms which were discovered from proofs in [13]. The experiments in the *Theorema* system are given in [10].

II. Proof–Based Synthesis

This section describes the problem of algorithm synthesis in the context of the binary trees and the proof-based synthesis techniques which we use.

A. Context

1) Notations: We consider binary trees over a totally ordered domain.

In our formulae there are two kinds of objects: domain objects which are tree members (usually denoted by lower-case letters – e.g. a, b, n), and binary trees (usually represented by upper-case letters – e.g. X, T, Y, Z). However the formulae do not indicate explicitly the types of the objects, but the type of each object is determined unambiguously by the context in which it occurs. Furthermore the meta–variables are starred (e.g., T^*, T_1^*, Z^*) and the Skolem constants have integer indices (e.g., X_0, X_1, a_0).

The ordering between tree elements is denoted by the usual $\leq$, and the ordering between a tree and an element is denoted by: $\preceq$ (e.g., $T \preceq z$ states that all the elements from the tree T are smaller or equal than the element z, $z \preceq T$ states that z is smaller or equal than all the elements from the tree T). We use two constructors for binary trees, namely: ε for the empty tree, and the triplet $\langle L, a, R\rangle$ for non-empty trees, where L and R are trees and a is the root element (e. g. $\langle \varepsilon, a, \varepsilon\rangle$ is a tree containing one single element).

A tree is a *sorted* (or *search*, or *ordered*) tree if it is either ε or of the form $\langle L, a, R\rangle$ such that i) $L \preceq a \preceq R$, and ii) L and R are sorted trees.

Functions: RgM, *LfM* have the following interpretations, respectively: $RgM[\langle L, n, R\rangle]$ (resp. $LfM[\langle L, n, R\rangle]$) returns the last (resp. first) visited element by traversing the tree $\langle L, n, R\rangle$ using the in-order traversal;

Predicates: $\approx$ and *IsSorted* with the following interpretations: $X \approx Y$ states that X and Y have the same elements with the same number of occurrences (but may have different structures), i.e., X is a *permutation* of Y; $IsSorted[X]$ states that X is a sorted tree.

All the statements used at object level in our experiments are formally just predicate logic formulae, and we call them differently depending on their role: a *definition* is given as an initial piece of the theory, considered to hold; a *property* is a logical consequence of the definitions; a *lemma* is a formula which we sometimes assume, and sometimes prove, depending of the current experiment scenario; and a *conjecture* is something we want to prove – typically in order to produce and algorithm. Not assumed properties and the conjectures are proved automatically by our system.

The following definitions and properties constitute the *knowledge base*.

Definition 1.

$$\underset{n,m,L,R,S}{\forall} \begin{pmatrix} RgM[\langle L, n, \varepsilon\rangle] = n \\ RgM[\langle L, n, \langle R, m, S\rangle\rangle] = RgM[\langle R, m, S\rangle] \end{pmatrix}$$

Definition 2.

$$\underset{n,m,L,R,S}{\forall} \begin{pmatrix} LfM[\langle \varepsilon, n, R\rangle] = n \\ LfM[\langle \langle L, n, R\rangle, m, S\rangle] = LfM[\langle L, n, R\rangle] \end{pmatrix}$$

Definition 3. $\underset{L,m,R}{\forall}$

$$\begin{pmatrix} IsSorted[\varepsilon] \\ (IsSorted[L] \wedge IsSorted[R] \wedge RgM[L] \leq m \leq LfM[R]) \Longleftrightarrow \\ IsSorted[\langle L, m, R\rangle] \end{pmatrix}$$

The functions *LfM* and *RgM* do not have a definition for the empty tree, however we assume:

Property 1. $\underset{m}{\forall}\Big(RgM[\varepsilon] \leq m \leq LfM[\varepsilon]\Big)$.

We also consider the following property which can be proved by induction from Definition 3:

Property 2. $\underset{z,T}{\forall}\Big(IsSorted[T] \Longrightarrow (T \preceq z \Longleftrightarrow RgM[T] \leq z)\Big)$

2) The Problem: We address the problem: having an element and a sorted binary tree, insert the element in the tree such that the result is sorted.

Algorithm synthesis is based on the proof of the following conjecture:

Conjecture 1. $\underset{\substack{n,X \\ IsSorted[X]}}{\forall} \underset{T}{\exists}\Big(\langle \varepsilon, n, X\rangle \approx T \wedge IsSorted[T]\Big)$

One can prove this conjecture in several ways. Each proof is different depending on the applied induction principle and the content of the knowledge base. Therefore, different algorithms are extracted from different proofs.

B. Induction Principle

In our experiments we use the following induction principle for proving P as unary predicate over binary trees.

Induction-1:

$$\Big(P[\varepsilon] \bigwedge \underset{n,L,R}{\forall}((P[L] \wedge P[R]) \Longrightarrow P[\langle L, n, R\rangle])\Big) \Longrightarrow \underset{X}{\forall} P[X]$$

The 'covering' property of the two induction conclusions $P[\varepsilon]$ and $P[\langle L, n, R\rangle]$ is satisfied since any binary tree is either ε or a triplet of the form $\langle L, n, R\rangle$. $P[L]$ and $P[R]$ are induction hypotheses attached to $P[\langle L, n, R\rangle]$, and it is very easy to see that their terms are smaller than the one of the induction conclusion, where the measure can be the height of the tree.

For instance, in order to synthesize the *Insert* algorithm as a function $F[n, X]$, we consider the output condition $O[n, X, T] : (\langle \varepsilon, n, X\rangle \approx T \wedge IsSorted[T])$. The corresponding

synthesis conjecture is:

$$\underset{\substack{n,X\\ \mathit{IsSorted}[X]}}{\forall} \underset{T}{\exists} O[n, X, T]$$

by taking $P[X]$ as $(\mathit{IsSorted}[X] \Longrightarrow \underset{T}{\exists} O[n, X, T])$.

We perform induction on variable X from the synthesis conjecture, thus the proof is structured as follows:

Base case: For arbitrary but fixed n (new constant), we prove $\underset{T}{\exists} O[n, \varepsilon, T]$. If the proof succeeds to find a ground witness $\Im_1[n, \varepsilon]$ (a term depending on n) such that $O[n, \varepsilon, \Im_1[n, \varepsilon]]$, then we know that $F[n, \varepsilon] = \Im_1[n, \varepsilon]$.

Step case: For arbitrary but fixed m, L and R (new constants), we prove

$$\mathit{IsSorted}[\langle L, m, R\rangle] \Longrightarrow \underset{T}{\exists} O[n, \langle L, m, R\rangle, T].$$

We assume as induction hypotheses:

$$\mathit{IsSorted}[L] \Longrightarrow \underset{T}{\exists} O[n, L, T]$$

and

$$\mathit{IsSorted}[R] \Longrightarrow \underset{T}{\exists} O[n, R, T],$$

which are Skolemized by introducing two new constants T_1 and T_2 for each existential T. The existentially quantified variable from the goal becomes the metavariable T^* (for which we need to find a substitution term). If the proof succeeds to find a witness $T^* = \Im_2[n, m, L, R, T_1, T_2]$ (term depending on n, m, L, R, T_1 and T_2), then we know that $F[n, \langle L, m, R\rangle] = \Im_2[n, m, L, R, F[n, L], F[n, R]]$ (T_1 and T_2 are replaced by $F[n, L]$ and $F[n, R]$, respectively).

The extracted algorithm from the proof has the following structure:

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

The process of algorithm extraction from the proof is also described in [9], but for a different conjecture.

C. Special Inference Rules and Strategies

Since the theory which we construct is expressed in first order logic, in principle the proofs can be performed by refutation and resolution, or by a search strategy like in Prolog. (In fact we also tried this approach in our earlier experiments.) However the proving process will be very long and memory consuming – in fact it will fail for more complex proofs for reasons of resource exhaustion. Additionally, it may happen that the resulting proof does not allow to extract an algorithm – because it uses non constructive inference steps. Therefore is very important to find efficient inference rules and strategies which allow the automatic proof in a reasonable time (any proof in our experiments does not overcome 5 seconds) and space, and furthermore to restrict the inference steps (in particular the creation of Skolem constants) in such a way that the produced proof is constructive (i. e. it allows the extraction of an algorithm).

The strategies and the inference rules which we apply were introduced by the authors in [9] and [13]. All these are based on the specific properties of our functions and predicates. We mention below only the special inference rules and strategies which are needed for the experiment which we describe in Section III. We detail **IR-3a** and **IR-3c**, which in [9] were only summarized, and **IR-5** and **S-3** which are not described neither in [9] nor in [13].

1) Special Inference Rules:

IR-1: *Generate Microatoms.* This rule is used in order to simplify both the goals and the assumptions. For assumptions it generates as many microatoms as possible and for the goal it generates a few microatoms that become conjuncts in the goal. Moreover, some of these microatoms will become conditional assumptions in the synthesized algorithm (see **IR-5** below).

We call *microatoms* those atoms whose arguments do not contain function symbols, except for few special ones – in the case of the current experiments we allow the functions *RgM* and *LfM* in microatoms.

Example: $\mathit{IsSorted}[\langle L, m, R\rangle]$ is transformed into $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{RgM}[L] \leq m \wedge m \leq \mathit{LfM}[R])$.

IR-2: *Eliminate-Ground-Formulae-from-Goal.* This rule increases the efficiency of proving by eliminating ground formulae from the goal.

Example: the assumption is: $L \preceq m$ and the goal is: $n \leq m \wedge L \preceq m \wedge m \leq \mathit{RgM}[R]$, the goal becomes: $n \leq m \wedge m \leq \mathit{RgM}[R]$.

The following two inference rules are a generalization of **IR-3** for replacing equivalent terms in goal (see [9]).

IR-3a: *Replace-Equivalent-Tree-Expression-in-Goal.* This rule constructs a different and equivalent tree expression. We illustrate on some examples.

Example-1: the assumption is: $\langle \varepsilon, n, L\rangle \approx T_1$ and the goal is: $\langle \varepsilon, n, \langle L, m, R\rangle\rangle \approx T^*$, then the goal is rewritten into: $\langle T_1, m, R\rangle \approx T^*$.

Example-2: the assumption is: $\langle \varepsilon, n, R\rangle \approx T_2$ and the goal is: $\langle \varepsilon, n, \langle L, m, R\rangle\rangle \approx T^*$, then the goal is rewritten into: $\langle L, m, T_2\rangle \approx T^*$.

IR-3c: *Replace-Equivalent-Atom-in-Goal.* This rule takes into account the interplay between the equivalence relation $\approx$, the orderings, and the functions *RgM*, *LfM* in order to perform similar replacements. We illustrate this rule on some examples.

Example-1: the assumptions are: $\langle \varepsilon, n, L\rangle \approx T_1$, $\mathit{IsSorted}[T_1]$ and the goal is: $\mathit{RgM}[T_1] \leq m$ then the goal is rewritten into: $n \leq m \wedge L \preceq m$.

Example-2: the assumptions are: $\langle \varepsilon, n, R\rangle \approx T_2$, $\mathit{IsSorted}[T_2]$ and the goal is: $m \leq \mathit{LfM}[T_2]$ then the goal is rewritten into: $m \leq n \wedge m \preceq R$.

All these transformation rules generate proof alternatives because they do not guarantee that the new goal is provable.

IR-5: *Simple-Goal-Conditional-Assumption.* When the goal is ground, no further simplification of it is possible, and the goal does not contain tree constants except inside the functions *RgM*, and *LfM*. Then this goal becomes a conditional assumption representing the condition attached to the corresponding branch of the synthesized algorithm, and the

current branch is considered successful (see also strategy **S-3**). The reason for the selection of such formulae as conditional assumptions is that they can be easily evaluated (an expression which does not contain tree expressions is evaluated in constant time, and the functions *RgM* and *LfM*, are evaluated in linear time).

2) Strategies:

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

S-3: *Case-Distinction.* The prover generates several proof branches, follows each branch in turn and produces 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: on one branch one obtains the condition $m \leq n$ and on another branch the condition $n \leq m$.

III. Experiments

In this subsection we describe the discovery of the *Insert* algorithm which is used by the sorting algorithms operating on binary trees. Some of the sorting algorithms are described in [13] and the experiments in the *Theorema* system (including the following one) are given in [11].

A. Synthesis of Insert

The following proof of Conjecture 1 constitute the synthesis of the auxiliary function *Insert*.

The prover automatically generates the proof of Conjecture 1 by applying **Induction-1** (on the second argument) and the specific inference rules and strategies from Subsection II-C. We describe below the most important steps of the proof. Note that the statement which has to be proven by induction is:

$$P[X] : \mathit{IsSorted}[X] \Longrightarrow (\underset{T}{\exists}(\langle \varepsilon, n, X\rangle \approx T \wedge \mathit{IsSorted}[T])).$$

Proof: After applying **Induction-1** and **S-1** to eliminate the existential quantifier, we get:

Base case: The witness found is $\{T^* \rightarrow \langle \varepsilon, n, \varepsilon\rangle\}$.

Induction step: We assume:

$$\mathit{IsSorted}[L] \Longrightarrow (\langle \varepsilon, n, L\rangle \approx T_1 \wedge \mathit{IsSorted}[T_1]) \quad (1)$$

$$\mathit{IsSorted}[R] \Longrightarrow (\langle \varepsilon, n, R\rangle \approx T_2 \wedge \mathit{IsSorted}[T_2]) \quad (2)$$

and we prove:

$$\begin{array}{c} \mathit{IsSorted}[\langle L, m, R\rangle] \Longrightarrow \\ (\langle \varepsilon, n, \langle L, m, R\rangle\rangle \approx T^* \wedge \mathit{IsSorted}[T^*]) \end{array} \quad (3)$$

We prove the right hand side of the above implication, by assuming the left hand side, which using **IR-1** is decomposed into:

$$\mathit{IsSorted}[L] \quad (4)$$

$$\mathit{IsSorted}[R] \quad (5)$$

$$\mathit{RgM}[L] \leq m \quad (6)$$

$$m \leq \mathit{LfM}[R] \quad (7)$$

By using Property 2 we extend these four assumptions with the new ones:

$$L \preceq m \quad (8)$$

$$m \preceq R \quad (9)$$

Using *modus ponens* from (4) and (5) by (1) and (2) we obtain:

$$\langle \varepsilon, n, L\rangle \approx T_1 \quad (10)$$

$$\mathit{IsSorted}[T_1] \quad (11)$$

$$\langle \varepsilon, n, R\rangle \approx T_2 \quad (12)$$

$$\mathit{IsSorted}[T_2] \quad (13)$$

The goal is:

$$\langle \varepsilon, n, \langle L, m, R\rangle\rangle \approx T^* \wedge \mathit{IsSorted}[T^*] \quad (14)$$

Since **IR-3a** can be applied on (14) in two different ways, the prover generates two alternatives:

Alternative-1: By applying **IR-3a** using the two assumptions (10) and (11), the goal is transformed into:

$$\langle T_1, m, R\rangle \approx T^* \wedge \mathit{IsSorted}[T^*] \quad (15)$$

Obtain substitution $\{T^* \longrightarrow \langle T_1, m, R\rangle\}$ and prove:

$$\mathit{IsSorted}[\langle T_1, m, R\rangle] \quad (16)$$

Apply **IR-1** and the goal becomes:

$$\mathit{IsSorted}[T_1] \wedge \mathit{IsSorted}[R] \wedge \mathit{RgM}[T_1] \leq m \wedge m \leq \mathit{LfM}[R] \quad (17)$$

Eliminate the first two conjuncts of the goal (apply **IR-2** using (11), (5)) and the new goal is:

$$\mathit{RgM}[T_1] \leq m \wedge m \leq \mathit{LfM}[R] \quad (18)$$

Apply **IR-3c** using (10) and (11) and the goal becomes:

$$n \leq m \wedge L \preceq m \wedge m \leq \mathit{LfM}[R] \quad (19)$$

Apply **IR-2** using (7) and (8) and the new goal is:

$$n \leq m \quad (20)$$

This goal fulfils the rule **IR-5** and thus it becomes the conditional assumption on this branch.

Alternative-2: By applying **IR-3a** using the two assumptions (12) and (13) the new goal is:

$$\langle L, m, T_2\rangle \approx T^* \wedge \mathit{IsSorted}[T^*] \quad (21)$$

Similar to the previous case and by using Property 2 we obtain the substitution $\{T^* \longrightarrow \langle L, m, T_2\rangle\}$ and the last goal is:

$$m \leq n \quad (22)$$

which becomes the conditional assumption on this branch. ■

The synthesized algorithm is extracted from the proof of Conjecture 1 as the definition of the following *Insert* function:

$$\underset{n,m,L,R}{\forall}\left(\begin{array}{l} \mathit{Insert}[n,\varepsilon] = \langle \varepsilon, n, \varepsilon\rangle \\ \mathit{Insert}[n,\langle L,m,R\rangle] = \\ \qquad \begin{cases} \langle \mathit{Insert}[n,L],m,R\rangle, & \text{if } n \leq m \\ \langle L,m,\mathit{Insert}[n,R]\rangle, & \text{otherwise} \end{cases} \end{array}\right)$$

Computations: We make some computations in *Theorema* with the obtained algorithm.

Input–1: Compute[*Insert*$[2,\langle\varepsilon,10,\varepsilon\rangle]$]

Output–1: $\langle\langle\varepsilon,2,\varepsilon\rangle,10,\varepsilon\rangle$

Input–2: Compute[*Insert*$[12,\langle\varepsilon,10,\varepsilon\rangle]$]

Output–2: $\langle\varepsilon,10,\langle\varepsilon,12,\varepsilon\rangle\rangle$

Input–3: Compute[*Insert*$[8,\langle\langle\varepsilon,9,\langle\varepsilon,11,\varepsilon\rangle\rangle,12,\varepsilon\rangle]$]

Output–3: $\langle\langle\langle\varepsilon,8,\varepsilon\rangle,9,\langle\varepsilon,11,\varepsilon\rangle\rangle,12,\varepsilon\rangle$

Input–4: Compute[Insert$[18,\langle\langle\varepsilon,9,\langle\varepsilon,12,\varepsilon\rangle\rangle,15,\varepsilon\rangle]$]

Output–4: $\langle\langle\varepsilon,9,\langle\varepsilon,12,\varepsilon,\rangle\rangle,15,\langle\varepsilon,18,\varepsilon\rangle\rangle$

B. Certification of Properties About the Synthesized Algorithm

According to [11], every insertion algorithm, denoted generically by *Ins* and supposed to be used in the synthesis of the merge algorithm on binary trees, should satisfy the following properties:

Property 3. $\underset{T}{\forall}\Big(\mathit{Ins}[n,T] \approx \langle\varepsilon,n,T\rangle\Big)$

Property 4. $\underset{T}{\forall}\Big(\mathit{IsSorted}[T] \Longrightarrow \mathit{IsSorted}[\mathit{Ins}[n,T]]\Big)$

For the particular case when *Ins* is instantiated by the *Insert* function, these properties become:

Property 5. $\underset{T}{\forall}\Big(\mathit{Insert}[n,T] \approx \langle\varepsilon,n,T\rangle\Big)$

Property 6. $\underset{T}{\forall}\Big(\mathit{IsSorted}[T] \Longrightarrow \mathit{IsSorted}[\mathit{Insert}[n,T]]\Big)$

Properties 5 and 6 have been mechanically certified using the Coq proof assistant [2].[1] *Insert* has been used to define a version of the merge algorithm, defined by the *Merge* function:

$$\underset{m,L,R,T}{\forall}\left(\begin{array}{l} \mathit{Merge}[\varepsilon,T] = T \\ \mathit{Merge}[\langle L,m,R\rangle,T] = \\ \qquad \mathit{Insert}[m,\mathit{Merge}[L,\mathit{Merge}[R,T]]] \end{array}\right)$$

for which the following crucial property has also been certified:

Property 7. $\underset{L,R}{\forall}\Big((\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R]) \Longrightarrow \mathit{IsSorted}[\mathit{Merge}[L,R]]\Big)$

During the certification process, several auxiliary lemmas have been used, as:

Lemma 1. $\underset{n,m,L,R}{\forall}\Big(\mathit{LfM}[\langle\mathit{Insert}[n,L],m,R\rangle] = n \ \vee \ \mathit{LfM}[\langle\mathit{Insert}[n,L],m,R\rangle] = \mathit{LfM}[\langle L,m,R\rangle]\Big)$

Lemma 2. $\underset{n,m,L,R}{\forall}\Big(\mathit{RgM}[\langle L,m,\mathit{Insert}[n,R]\rangle] = n \ \vee \ \mathit{RgM}[\langle L,m,\mathit{Insert}[n,R]\rangle] = \mathit{RgM}[\langle L,m,R\rangle]\Big)$

IV. Conclusions and Further Work

This experiment illustrates the usefulness of the proof–based approach to the automatic synthesis of a nontrivial algorithm. The success of the approach is the result of the application of inference rules and strategies which are specific to the domain of binary trees – in fact some of them are also applicable to lists and to other domains. The principles which are more general include: induction principles based on the definition of the domain, the treatment of quantifiers using Skolem constants and metavariables, decomposition into microatoms, and simple–goal conditional assumptions.

In the context of the problem of sorting of binary trees, the experiment also contributed to the illustration and understanding of the process of theory exploration: the construction of the appropriate theory in parallel with the attempt to define the specification of the desired function and to prove the necessary properties.

The results of this experiment opens the way for the exploration of more complex problems, for instance by using a combination of the theory of lists with the theory of binary trees.

[1]The full Coq script is available at: http://web.info.uvt.ro/~idramnesc/LATA2016/coq.v

References

[1] D. Basin, Y. Deville, P. Flener, A. Hamfelt, and J. F. Nilsson. Synthesis of Programs in Computational Logic. In *Program Development in Computational Logic*, pages 30–65. Springer, 2004.

[2] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*, volume XXV of *Texts in Theoretical Computer Science. An EATCS*. Springer, 2004.

[3] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 4(4):470–504, 2006.

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

[5] A. Bundy, L. Dixon, J. Gow, and J. Fleuriot. Constructing Induction Rules for Deductive Synthesis Proofs. *Electron. Notes Theor. Comput. Sci.*, 153:3–21, March 2006.

[6] A. Bundy, A. Smaill, and G. Wiggins. The Synthesis of Logic Programs from Inductive Proofs. In J. W. Lloyd, editor, *Computational Logic: Symposium Proc.*, pages 135–149. Springer, 1990.

[7] Y. Deville and K. K. Lau. Logic Program Synthesis. *J. Log. Program.*, 19/20:321–350, 1994.

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

[9] I. Dramnesc, T. Jebelean, and S. Stratulat. Combinatorial techniques for proof-based synthesis of sorting algorithms. In *SYNASC 2015: Proceedings of the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 137–144, 2015.

[10] I. Dramnesc, T. Jebelean, and S. Stratulat. Synthesis of Some Algorithms for Trees: Experiments in Theorema. Technical Report 15-04, RISC Report Series, University of Linz, Austria, 2015.

[11] I. Dramnesc, T. Jebelean, and S. Stratulat. Synthesis of Some Algorithms for Trees: Experiments in Theorema. Technical Report 15-04, RISC Report Series, Johannes Kepler University, Linz, Austria, 2015.

[12] I. Dramnesc, T. Jebelean, and S. Stratulat. Theory exploration of binary trees. In *13th IEEE International Symposium on Intelligent Systems and Informatics (SISY 2015)*, pages 139–144. IEEE Publishing, 2015.

[13] I. Dramnesc, T. Jebelean, and S. Stratulat. Proof-based Synthesis of Sorting Algorithms for Trees. In *Proceedings of LATA 2016: 10th International Conference on Language and Automata Theory and Applications*, volume LNCS 9618. Springer, 2016.

[14] P. Flener. Achievements and Prospects of Program Synthesis. In *Computational Logic: Logic Programming and Beyond*, pages 310–346, 2002.

[15] D. E. Knuth. *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*. Addison-Wesley Longman Publishing Co.,Inc., Redwood City, CA, USA, 1998.

[16] I. Kraan, D. A. Basin, and A. Bundy. Middle-Out Reasoning for Synthesis and Induction. *Journal Of Automated Reasoning*, 16(1-2):113–145, 1996.

[17] S. Wolfram. *The Mathematica Book*. Wolfram Media Inc., 2003.