Theory Exploration of Binary Trees

Isabela Dramnesc, Tudor Jebelean, Sorin Stratulat

Theory Exploration of 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*—The construction of a theory for binary trees is presented, based on the systematic exploration of the properties necessary for the proof-based synthesis and certification of sorting algorithms for binary trees. The process is computer supported, being realised in the frame of the *Theorema* system, with some additional proofs in Coq required for algorithm certification. The result of the exploration consists in 11 definitions, 3 axioms, and more than 200 properties. Also, more than 5 algorithms for sorting binary trees are generated.**

***Keywords*—automated reasoning, theory exploration, binary trees, Theorema, Coq**

I. Introduction

A theory of binary trees over ordered domains is constructed, by systematic exploration in the context of proof-based synthesis of sorting algorithms. The theory contains basic definitions and axioms, properties inferred from these, as well as propositions and conjectures relevant to the problem of sorting. The theory is represented in the *Theorema* system [5], and the proofs are also performed automatically in the frame of this system. Additionally properties aiming to certify synthesized sorting algorithms in the Coq system [1] have been mechanically provided and proved. In the authors' view, a mathematical theory consists in a *knowledge base* (axioms, definitions, propositions, etc.) – that is a collection of logical formulae, over a certain signature (function, predicate and constant symbols).

In general, mathematical theories are built incrementally starting from a set of axioms, definitions, and propositions, which are checked and proved. All the steps that mathematicians make for exploring mathematical theories like defining new notions, checking and proving propositions about the notions, computing, solving, defining problems and their solutions, can be supported by specific computer systems. The *Theorema* system has been used, because it supports a notation similar to the one used in practical mathematics, and also provides a framework in which the natural style proof methods can be designed and implemented, moreover producing proofs in natural language, similar to human produced proofs. For comparison and further check, some properties are also formalized and proved in the Coq system.

In [3] the author explains the processes of bottom-up and top-down exploration of theories. In the process of bottom-up exploration one starts from a set of axioms and generates new knowledge base by introducing definitions and propositions. In the process of top-down exploration one starts from a problem and adds notions and properties of the notions to the knowledge base. The author also points out the advantages and the disadvantages of these two processes of theory exploration. In this paper these two directions of theory exploration have been used. First, the most obvious axioms and definitions of the notions which are necessary for the sorting of binary trees have been introduced, and their elementary properties have been proved. After that the conjectures necessary for the proof-based synthesis have been attempted to prove, and then various properties and notions which are still necessary have been identified. This leads to a new bottom-up exploration in which the interaction between the old and the new notions are investigated, until all necessary properties are discovered and proved.

A. Applications

The immediate purpose of the theory development here is the automated synthesis of algorithms on binary trees. This approach opens the way for the effective automation of proofs, of the exploration of theory and of the synthesis of the algorithms applied on binary trees. This case study in theory exploration can be also used in teaching, especially because it is completely supported by the *Theorema* system (`https://www.risc.jku.at/research/theorema/software`). However, other applications are obviously possible, for instance reasoning about binary trees in the context of program verification.

B. Related work

The scheme-based model for theory exploration in the *Theorema* system, introduced in [4] by Bruno Buchberger, is applied on natural numbers in [6].

The set theory is described in [16], and [15]. Sets are normalized in [2] in order to obtain sorted lists without duplications. In the context of proof–based algorithm synthesis in [8], and [11] the authors represent sets as sorted lists without duplication, called *monotone lists*, and in [10] the authors explore the corresponding theories of sets and monotone lists.

The exploration process of the theory of lists, in natural style, and related to a complex application like automated synthesis, see [12], is described in [9].

In contrast to all these studies, in the present work the theory of binary trees is explored, which is carried out in parallel with the process of proof–based synthesis of some sorting algorithms.

In classical approaches (see e.g. [14]) the problem of sorting of trees is not investigated.

II. Basic Approach

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: $\underset{X}{\forall}$ ("for all X") is the universal quantifier, while $\underset{X}{\exists}$ ("exists X") is the existential quantifier.

The theory applies to binary trees and to elements of those (which have no type but have a total ordering), however the type of objects are not explicitly present in the formulae. This is not a problem because the predicate and function symbols are not overloaded. For readability, lower-case letters (e.g., a, b, n) represent tree elements, and upper-case letters (e.g., X, T, Y, Z) represent trees. Usually, 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).

Binary trees whose nodes are labeled with elements from a totally ordered domain have been considered, the ordering relation being denoted by the usual $\leq$ ("less or equal"). This ordering is extended to trees as follows. 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 equal than the element z; $z \preceq T$ states that z is smaller equal than all the elements from the tree T). The ordering between trees is $\lessdot$ (e.g., $L \lessdot R$ states that all the elements from L are smaller than all the elements from R).

Two constructors for binary trees have been used, 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.

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) a is greater or equal than any element of L and smaller or equal than any element of R, and ii) L and R are sorted trees.

The signature of our theory contains furthermore the following predicates and functions:

Predicates: $\approx$ and *IsSorted* have the following interpretations, respectively: $X \approx Y$ 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.

Functions: *RgM*, *LfM*, *Concat*, *Insert*, *Merge* have the following interpretations: *RgM*$[\langle L, n, R\rangle]$ returns the last visited element by traversing the tree $\langle L, n, R\rangle$ using the in-order traversal; *LfM*$[\langle L, n, R\rangle]$ returns the first element by traversing the tree $\langle L, n, R\rangle$ using the in-order traversal; *Concat*$[X, Y]$ concatenates X with Y (namely, when X is of the form $\langle L, n, R\rangle$ adds Y as a right subtree of the element *RgM*$[\langle L, n, R\rangle]$); *Insert*$[n, X]$ inserts an element n in a tree X (if X is sorted, then the result is also sorted); *Merge*$[X, Y]$ combines trees X and Y into a new tree (if X, Y are sorted then the result is also sorted).

A. Reasoning

As the theory exploration is performed in the context of proof-based synthesis, a special prover has been implemented in frame of the *Theorema* system, which has specific inference rules for binary trees. The illustration of the inference rules and of the induction principles presented in this subsection is similar to the one from [10], but they are adapted and modified for binary trees.

The prover has mainly the following types of inference rules:

Rewriting rules: rewrite using definitions, by replacing equals by equals.

Domain specific inference rules: which result from lifting some properties from the knowledge base to the inference level. These rules are useful because they do not generate alternative branches in proofs and thus one avoids the search space explosion.

Matching and unification: The most general case is matching a conjunctive goal with the conclusion of an universal assumed implication. E.g.: if the assumption is $(ExprL \Longrightarrow ExprR)$ and the goal is $G1 \wedge G2$ it first tries to do matching on $G1$ with $ExprR$ (or on $G2$ with $ExprR$) of the assumed implication and if the matching is done with the substitution σ, then it is sufficient to prove $(ExprL)\sigma \wedge G2$ (or $(ExprL)\sigma \wedge G1$).

A special situation is when variables become meta-variables: if after matching no substitution is found for a variable, then that variable becomes a meta-variable for which a substitution term is needed using unification. This situation is also described in [7].

Induction principles: lifted to the inference level, they are adapted to prove properties about binary trees.

In the experiments the following induction principles have been used for proving properties $P[\overline{X}]$, where $\overline{X}$ is a vector of variables and some of them have as sort binary trees. They are (direct or indirect) *term-based* instances of the Noetherian induction principle [17] and are represented using *induction schemas*.

When $\overline{X}$ has only one variable of binary tree sort:

Induction-1:

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

Induction-2:

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

Induction-3:

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

When $\overline{X}$ has two variables of binary tree sort:

Induction-4:

$$\Big(P[\varepsilon, \varepsilon] \bigwedge \underset{b,C,D}{\forall}((P[\varepsilon, C] \wedge P[\varepsilon, D]) \Longrightarrow P[\varepsilon, \langle C, b, D\rangle]) \bigwedge \underset{a,A,B}{\forall}((P[A, \varepsilon] \wedge P[B, \varepsilon]) \Longrightarrow P[\langle A, a, B\rangle, \varepsilon]) \bigwedge$$

$$\underset{a,b,A,B,C,D}{\forall}((P[\langle A,a,B\rangle,C]\wedge P[\langle A,a,B\rangle,D]\wedge P[A,\langle C,b,D\rangle]\wedge P[B,\langle C,b,D\rangle])\Longrightarrow P[\langle A,a,B\rangle,\langle C,b,D\rangle])\Big)\Longrightarrow \underset{X,Y}{\forall}P[X,Y]$$

When $\overline{X}$ has one variable of element sort and one variable of binary tree sort:

Induction-5:

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

III. Theory Exploration

This section presents the formal definitions of the functions and predicates and their properties. As usual, the free variables occurring in these formulae are implicitly universally quantified.

A. Basic Notions

1) The relation "$\leq$" between elements:

P-1. $(a\leq b\wedge b\leq c)\Longrightarrow a\leq c$

2) The relation "$\preceq$" between an element and a tree:

Definition 1. $\underset{n,L}{\forall}\Big(\underset{\mathit{Member}[m,L]}{\forall}\Big(n\leq m\Big)\Longleftrightarrow n\preceq L\Big)$

One adds the following properties in the knowledge base:

P-2. $m\preceq\varepsilon\ \wedge\ \varepsilon\preceq m$

P-3. $(m\preceq L\wedge m\leq n\wedge m\preceq R)\Longrightarrow m\preceq\langle L,n,R\rangle$

P-4. $(a\leq b\wedge b\preceq L)\Longrightarrow a\preceq L$

3) The relation "$\underline{\ll}$ " between trees:

Definition 2. $\underset{L,R}{\forall}\Big(L\underline{\ll}R\Longleftrightarrow\underset{\substack{\mathit{Member}[m,L]\\ \mathit{Member}[n,R]}}{\forall}\Big(m\leq n\Big)\Big)$

P-5. $L\underline{\ll}\varepsilon$

P-6. $\varepsilon\underline{\ll}L$

P-7. $(L\underline{\ll}S\wedge n\preceq S\wedge R\underline{\ll}S)\Longleftrightarrow(\langle L,n,R\rangle\underline{\ll}S)$

P-8. $(L\underline{\ll}S\wedge R\underline{\ll}S\ \wedge L\underline{\ll}T\wedge R\underline{\ll}T\wedge m\preceq S\wedge m\preceq T\wedge m\leq n\wedge L\preceq n\wedge R\preceq n)\Longleftrightarrow(\langle L,m,R\rangle\underline{\ll}\langle S,n,T\rangle)$

P-9. $(L\underline{\ll}R\wedge R\underline{\ll}S)\Longrightarrow L\underline{\ll}S$

P-10. $(L\preceq n\wedge n\preceq R)\Longrightarrow L\underline{\ll}R$

P-11. $(m\preceq L\wedge L\underline{\ll}R)\Longrightarrow m\preceq R$

P-12. $L\underline{\ll}R\Longleftrightarrow\underset{\mathit{Member}[m,L]}{\forall}\Big(m\preceq R\Big)$

P-13. $L\underline{\ll}R\Longleftrightarrow\underset{\mathit{Member}[m,R]}{\forall}\Big(L\preceq m\Big)$

4) The right most element:

Definition 3.

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

5) The left most element:

Definition 4.

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

Remark. The functions *LfM* and *RgM* do not have a definition for the empty tree, however the following axiom has been assumed:

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

6) The 'IsSorted' predicate:

Definition 5. $\underset{L,m,R}{\forall}$

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

The following properties have been added in the knowledge base:

P-14. $\mathit{IsSorted}[T]\Longrightarrow(T\preceq z\Longleftrightarrow\mathit{RgM}[T]\leq z)$

P-15. $\mathit{IsSorted}[T]\Longrightarrow(z\preceq T\Longleftrightarrow z\leq\mathit{LfM}[T])$

P-16. $(\mathit{IsSorted}[L]\wedge\mathit{IsSorted}[R]\wedge L\preceq n\wedge n\preceq R\wedge L\underline{\ll}R)\Longleftrightarrow\mathit{IsSorted}[\langle L,n,R\rangle]$

7) The 'Member' predicate:

Axiom 2. $\underset{n}{\forall}\neg\Big(\mathit{Member}[n,\varepsilon]\Big)$

Definition 6. $\underset{a,b,L,R}{\forall}\Big(\mathit{Member}[a,\langle L,b,R\rangle]\Longleftrightarrow((a=b)\vee\mathit{Member}[a,L]\ \vee\mathit{Member}[a,R])\Big)$

P-17. $\mathit{Member}[a,L]\Longrightarrow\mathit{Member}[a,\langle L,b,R\rangle]$

P-18. $\mathit{Member}[a,R]\Longrightarrow\mathit{Member}[a,\langle L,b,R\rangle]$

P-19. $\mathit{Member}[a,\langle L,a,R\rangle]$

8) The permutation of a tree:

Definition 7.

$$\underset{T,T'}{\forall}\begin{pmatrix}((\underset{x}{\forall}(\mathit{Member}[x,T]\Longrightarrow(\mathit{Member}[x,T']\wedge\mathit{NbOcc}[x,T]=\mathit{NbOcc}[x,T']))\ \wedge\\ \wedge\ (\underset{x}{\forall}(\mathit{Member}[x,T']\Longrightarrow(\mathit{Member}[x,T]\wedge\mathit{NbOcc}[x,T]=\mathit{NbOcc}[x,T'])))\\ \Longleftrightarrow T\approx T'\end{pmatrix}$$

where *NbOcc* is the function which returns the number of occurrences of an element into a tree. The definition is not used explicitly in the proofs.

Additionally, we have considered the following properties:

P-20. $(L\approx R)\Longrightarrow(\mathit{NbOcc}[a,L]=\mathit{NbOcc}[a,R])$

P-21. $(L\approx R)\Longrightarrow(\mathit{Member}[a,L]\Longleftrightarrow\mathit{Member}[a,R])$

P-22. $\langle L,n,R\rangle\approx\langle R,n,L\rangle$

Note also that the $\approx$ relation is an equivalence (this is used implicitly by the prover).

B. Auxiliary functions

These functions are used in the sorting algorithms.

1) The concatenation of trees:

Definition 8.
$$\forall_{n,L,R,S} \left(\begin{array}{c} \mathit{Concat}[\varepsilon, R] = R \\ \mathit{Concat}[\langle L, n, R\rangle, S] = \langle L, n, \mathit{Concat}[R, S]\rangle \end{array} \right)$$

A first simple property which can be proven inductively from Definition 8 is the following:

P-23. $\mathit{Concat}[L, \varepsilon] = L$

Other properties:

P-24. $(L \lessdot R \wedge S \lessdot R) \Longleftrightarrow \mathit{Concat}[L, S] \lessdot R$

P-25. $(L \lessdot R \wedge L \lessdot S) \Longleftrightarrow L \lessdot \mathit{Concat}[R, S]$

P-26. $(n \preceq L \wedge n \preceq R) \Longleftrightarrow n \preceq \mathit{Concat}[L, R]$

P-27. $(L \preceq n \wedge R \preceq n) \Longleftrightarrow \mathit{Concat}[L, R] \preceq n$

P-28. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge L \lessdot R) \Longleftrightarrow$
$\mathit{IsSorted}[\mathit{Concat}[L, R]]$

P-29. $\mathit{Concat}[L, R] \approx \mathit{Concat}[R, L]$

2) The insertion of an element in a tree. If the tree is sorted, then the resulted tree is sorted:

Definition 9. $\forall_{n,m,L,R}$
$$\left(\begin{array}{c} \mathit{Insert}[n, \varepsilon] = \langle \varepsilon, n, \varepsilon\rangle \\ \mathit{Insert}[n, \langle L, m, R\rangle] = \begin{cases} \langle L, m, \mathit{Insert}[n, R]\rangle, & \text{if } m \leq n \\ \langle \mathit{Insert}[n, L], m, R\rangle, & \text{otherwise} \end{cases} \end{array} \right)$$

P-30. $\langle L, n, R\rangle \approx \mathit{Insert}[n, \mathit{Concat}[L, R]]$

P-31. $\langle L, n, R\rangle \approx \mathit{Insert}[n, \mathit{Concat}[R, L]]$

P-32. $\mathit{IsSorted}[X] \Longrightarrow \mathit{IsSorted}[\mathit{Insert}[n, X]]$

3) The 'Merge' operation between two trees. If the two trees are sorted, then the resulted tree is sorted:

Definition 10. $\forall_{n,L,R,S}$
$$\left(\begin{array}{c} \mathit{Merge}[\varepsilon, R] = R \\ \mathit{Merge}[\langle L, n, R\rangle, S] = \mathit{Insert}[n, \mathit{Merge}[L, \mathit{Merge}[R, S]]] \end{array} \right)$$

P-33. $\mathit{Merge}[T, \varepsilon] \approx T$

P-34. $\mathit{Merge}[\varepsilon, T] \approx T$

Other properties added to the knowledge base are:

P-35. $(L \lessdot R \wedge L \lessdot S) \Longleftrightarrow L \lessdot \mathit{Merge}[R, S]$

P-36. $(L \lessdot S \wedge R \lessdot S) \Longleftrightarrow \mathit{Merge}[L, R] \lessdot S$

P-37. $(L \preceq n \wedge R \preceq n) \Longleftrightarrow \mathit{Merge}[L, R] \preceq n$

P-38. $(n \preceq L \wedge n \preceq R) \Longleftrightarrow n \preceq \mathit{Merge}[L, R]$

P-39. $\langle L, n, R\rangle \approx \mathit{Insert}[n, \mathit{Merge}[L, R]]$

P-40. $\langle L, n, R\rangle \approx \mathit{Insert}[n, \mathit{Merge}[R, L]]$

P-41. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R]) \Longrightarrow \mathit{IsSorted}[\mathit{Merge}[L, R]]$

P-42. $\mathit{Merge}[L, R] \approx \mathit{Merge}[R, L]$

C. Useful properties for proofs

1) Using multisets: Since the relation $\approx$ is an equivalence, reasoning about it often reduces to reasoning about the multiset of elements of the trees, which furthermore reduces to reasoning about the multiset of symbols occurring in an expression which represents a tree. This is because most of the functions which operate on trees combine the multisets of elements of their arguments (*Concat, Merge, Insert*, and the constructor $\langle \ldots \rangle$).

As an illustration of this principle 6 properties have been listed below which correspond to some of the permutations of the tree $\langle L, n, \mathit{Merge}[R, S]\rangle$. These properties can be generated automatically in the *Theorema* system. The automation mechanism consists in: having an expression, obtain its multiset of symbols, then generate the permutations of the multiset, and then build an equivalent expression by using the constructor $\langle\rangle$, and the functions *Concat*, *Insert*, and *Merge*.

P-43. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \mathit{Insert}[n, \mathit{Merge}[L, \mathit{Merge}[R, S]]]$

P-44. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \mathit{Insert}[n, \mathit{Merge}[\mathit{Merge}[L, R], S]]$

P-45. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \mathit{Insert}[n, \mathit{Concat}[L, \mathit{Merge}[R, S]]]$

P-46. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \mathit{Insert}[n, \mathit{Merge}[\mathit{Concat}[L, R], S]]$

P-47. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \langle L, n, \mathit{Concat}[R, S]\rangle$

P-48. $\langle L, n, \mathit{Merge}[R, S]\rangle \approx \langle \mathit{Merge}[L, R], n, S\rangle$

Similarly, one obtains more than 100 properties corresponding to the permutations of the trees: $\langle L, n, \mathit{Concat}[R, S]\rangle$, $\langle \mathit{Merge}[L, R], n, S\rangle$, and $\langle \mathit{Concat}[L, R], n, S\rangle$. These properties can be extended for more complex trees.

2) Decomposition into microatoms: During the proof development it is sometimes useful to replace an atom whose arguments include tree expressions into several atoms whose arguments consist only of variables or constants – thus no composite terms (the latter have been called *microatoms*). The decomposition can be done both forward (generate microatoms as consequences of an assumption) and backward (replace an atom in the goal by the microatoms whose consequence it is). Some properties which are necessary for such decompositions are the following:

P-49. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S]) \Longrightarrow$
$\mathit{IsSorted}[\mathit{Insert}[n, \mathit{Merge}[L, \mathit{Merge}[R, S]]]]$

P-50. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S]) \Longrightarrow$
$\mathit{IsSorted}[\mathit{Insert}[n, \mathit{Merge}[\mathit{Merge}[L, R], S]]]$

The proofs of Properties 49 and 50 come directly from Properties 32, 41.

P-51. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S] \wedge L \lessdot R \wedge$
$L \lessdot S) \Longrightarrow \mathit{IsSorted}[\mathit{Insert}[n, \mathit{Concat}[L, \mathit{Merge}[R, S]]]]$

P-52. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S] \wedge L \lessdot R)$
$\Longrightarrow \mathit{IsSorted}[\mathit{Insert}[n, \mathit{Merge}[\mathit{Concat}[L, R], S]]]$

The proof of Property 52 comes directly from Properties 25, 28, 32, and 41.

P-53. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S] \wedge L \preceq n \wedge n \preceq$

$$R \wedge n \preceq S \wedge R \lessdot S) \Longrightarrow \mathit{IsSorted}[\langle L, n, \mathit{Concat}[R, S]\rangle]$$

Property 53 is easily proved using Properties 16, 25, 26, 28, and 9.

P-54. $(\mathit{IsSorted}[L] \wedge \mathit{IsSorted}[R] \wedge \mathit{IsSorted}[S] \wedge L \preceq n \wedge$

$$R \preceq n \wedge n \preceq S) \Longrightarrow \mathit{IsSorted}[\langle \mathit{Merge}[L, R], n, S\rangle]$$

Property 54 is proved by Properties 16, 41, 37, and 36.

Similarly, one obtains numerous properties which correspond to the sorted permutations of the trees $\langle L, n, \mathit{Concat}[R, S]\rangle$, $\langle \mathit{Merge}[L, R], n, S\rangle$, and $\langle \mathit{Concat}[L, R], n, S\rangle$.

IV. Sorting algorithms

The final goal of the theory exploration case study is the generation of sorting algorithms. The following sorting algorithms are new and they have been discovered automatically from proofs by the new prover which the authors implemented in the *Theorema* system [5] (see, e.g., [5]) which is itself implemented in Mathematica [18]. The method and the algorithms extraction are presented in other authors' paper [13].

Each algorithm has been extracted from the proof of the conjecture formalizing the statement "for each binary tree X there exists a sorted binary tree Y having the same elements as X". (Different proofs yield different algorithms.) To each algorithm corresponds the theorem that its result is sorted.

Sort-1: The following algorithm uses the functions *LfM*, *RgM*, *Insert* and *Merge*.

Algorithm 1.

$$\underset{n,L,R}{\forall}\left(\begin{array}{c} F_1[\varepsilon] = \varepsilon \\ F_1[\langle \varepsilon, n, \varepsilon\rangle] = \langle \varepsilon, n, \varepsilon\rangle \\ F_1[\langle \varepsilon, n, R\rangle] = \begin{cases} \langle \varepsilon, n, F_1[R]\rangle, & \textit{if } n \le \mathit{LfM}[F_1[R]] \\ \langle F_1[R], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_1[R]] \le n \\ \mathit{Insert}[n, F_1[R]], & \textit{otherwise} \end{cases} \\ F_1[\langle L, n, \varepsilon\rangle] = \begin{cases} \langle \varepsilon, n, F_1[L]\rangle, & \textit{if } n \le \mathit{LfM}[F_1[L]] \\ \langle F_1[L], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_1[L]] \le n \\ \mathit{Insert}[n, F_1[L]], & \textit{otherwise} \end{cases} \\ F_1[\langle L, n, R\rangle] = \mathit{Insert}[n, \mathit{Merge}[F_1[L], F_1[R]]] \end{array}\right)$$

Sort-2: The following algorithm uses the functions *LfM*, *RgM*, *Insert*, *Concat*, and *Merge*.

Algorithm 2.

$$\underset{n,L,R}{\forall}\left(\begin{array}{c} F_2[\varepsilon] = \varepsilon \\ F_2[\langle L, n, \varepsilon\rangle] = \begin{cases} \langle F_2[L], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_2[L]] \le n \\ \langle \varepsilon, n, F_2[L]\rangle, & \textit{if } n \le \mathit{LfM}[F_2[L]] \\ \mathit{Insert}[n, F_2[L]], & \textit{otherwise} \end{cases} \\ F_2[\langle L, n, R\rangle] = \mathit{Merge}[F_2[\langle L, n, \varepsilon\rangle], F_2[R]] \end{array}\right)$$

Sort-3: The following algorithm is similar to F_1, excepting the last case where F_3 uses the functions *Insert* and *Concat*.

Algorithm 3.

$$\underset{n,L,R}{\forall}\left(\begin{array}{c} F_3[\varepsilon] = \varepsilon \\ F_3[\langle \varepsilon, n, \varepsilon\rangle] = \langle \varepsilon, n, \varepsilon\rangle \\ F_3[\langle \varepsilon, n, R\rangle] = \begin{cases} \langle \varepsilon, n, F_3[R]\rangle, & \textit{if } n \le \mathit{LfM}[F_3[R]] \\ \langle F_3[R], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_3[R]] \le n \\ \mathit{Insert}[n, F_3[R]], & \textit{otherwise} \end{cases} \\ F_3[\langle L, n, \varepsilon\rangle] = \begin{cases} \langle F_3[L], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_3[L]] \le n \\ \langle \varepsilon, n, F_3[L]\rangle, & \textit{if } n \le \mathit{LfM}[F_3[L]] \\ \mathit{Insert}[n, F_3[L]], & \textit{otherwise} \end{cases} \\ F_3[\langle L, n, R\rangle] = \mathit{Insert}[n, F_3[\mathit{Concat}[L, R]]] \end{array}\right)$$

Sort-4: This algorithm is similar to F_1, excepting the last branch where F_4 has three cases.

Algorithm 4.

$$\underset{n,L,R}{\forall}\left(\begin{array}{c} F_4[\varepsilon] = \varepsilon \\ F_4[\langle \varepsilon, n, \varepsilon\rangle] = \langle \varepsilon, n, \varepsilon\rangle \\ F_4[\langle L, n, \varepsilon\rangle] = \begin{cases} \langle F_4[L], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_4[L]] \le n \\ \langle \varepsilon, n, F_4[L]\rangle, & \textit{if } n \le \mathit{LfM}[F_4[L]] \\ \mathit{Insert}[n, F_4[L]], & \textit{otherwise} \end{cases} \\ F_4[\langle \varepsilon, n, R\rangle] = \begin{cases} \langle \varepsilon, n, F_4[R]\rangle, & \textit{if } n \le \mathit{LfM}[F_4[R]] \\ \langle F_4[R], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_4[R]] \le n \\ \mathit{Insert}[n, F_4[R]], & \textit{otherwise} \end{cases} \\ F_4[\langle L, n, R\rangle] = \begin{cases} \langle F_4[L], n, F_4[R]\rangle, \\ \qquad \textit{if } (\mathit{RgM}[F_4[L]] \le n \wedge \\ \qquad n \le \mathit{LfM}[F_4[R]]) \\ \langle F_4[L], n, F_4[R]\rangle, \\ \qquad \textit{if } (\mathit{RgM}[F_4[R]] \le n \wedge \\ \qquad n \le \mathit{LfM}[F_4[L]]) \\ \mathit{Insert}[n, \mathit{Merge}[F_4[L], F_4[R]]], \ \textit{ow.} \end{cases} \end{array}\right)$$

Sort-5: This algorithm is similar to F_3, excepting the last case where F_5 has three branches.

Algorithm 5.

$$\underset{n,L,R}{\forall}\left(\begin{array}{c} F_5[\varepsilon] = \varepsilon \\ F_5[\langle \varepsilon, n, \varepsilon\rangle] = \langle \varepsilon, n, \varepsilon\rangle \\ F_5[\langle \varepsilon, n, R\rangle] = \begin{cases} \langle \varepsilon, n, F_3[R]\rangle, & \textit{if } n \le \mathit{LfM}[F_5[R]] \\ \langle F_5[R], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_5[R]] \le n \\ \mathit{Insert}[n, \mathit{Merge}[\varepsilon, F_5[R]]], & \textit{ow.} \end{cases} \\ F_5[\langle L, n, \varepsilon\rangle] = \begin{cases} \langle F_5[L], n, \varepsilon\rangle, & \textit{if } \mathit{RgM}[F_5[L]] \le n \\ \langle \varepsilon, n, F_5[L]\rangle, & \textit{if } n \le \mathit{LfM}[F_5[L]] \\ \mathit{Insert}[n, \mathit{Merge}[F_5[R], \varepsilon]], & \textit{ow.} \end{cases} \\ F_5[\langle L, n, R\rangle] = \begin{cases} \langle F_5[L], n, F_5[R]\rangle, & \textit{if } \mathit{RgM}[F_5[L]] \le n \wedge \\ & \quad n \le \mathit{LfM}[F_5[R]] \\ \langle F_5[R], n, F_5[L]\rangle, & \textit{if } \mathit{RgM}[F_5[R]] \le n \wedge \\ & \quad n \le \mathit{LfM}[F_5[L]] \\ \mathit{Insert}[n, F_5[\mathit{Concat}[L, R]]], & \textit{ow.} \end{cases} \end{array}\right)$$

Other sorting algorithms are generated (by generating the permutations of a triplet of the form $\langle T_1, n, T_2\rangle$ and by using the functions *Insert*, *Concat*, *Merge*, and the constructor $\langle\rangle$), but they are not shown here due to lack of space.

V. Additional certification of properties and sorting algorithms

The correctness of the approach presented in this paper is guaranteed by the way the synthesized algorithms are built. However, the generated theory and the implemented inference rules used for synthesizing algorithms are error-prone and can be tested by certifying that the synthesized algorithms are indeed sorting algorithms. The certification effort is important and mechanized assistance is recommended. For example, the *Sort-1* algorithm has been successfully certified using Coq [1]. See the Coq script at `http://web.info.uvt.ro/~idramnesc/SISY2015/coq.v`. Even if few properties are shared between the certification and synthesis proofs, the certification proof is built using different inference rules and proof scripts, in a less automatic way and requiring additional properties:

P-55. $\mathit{IsSorted}[\langle L, n, R\rangle] \Longrightarrow \mathit{IsSorted}[L]$

P-56. $\mathit{IsSorted}[\langle L, n, R\rangle] \Longrightarrow \mathit{IsSorted}[R]$

P-57. $\mathit{IsSorted}[\langle T_1, n, \langle L, m, R\rangle\rangle] \Longrightarrow n \leq \mathit{LfM}[\langle L, m, R\rangle]$

P-58. $\mathit{IsSorted}[\langle\langle L, m, R\rangle, n, T_2\rangle] \Longrightarrow \mathit{RgM}[\langle L, m, R\rangle] \leq n$

P-59. $\mathit{IsSorted}[\langle L, m, R\rangle] \wedge \mathit{RgM}[\langle L, m, R\rangle] \leq n \Longrightarrow \mathit{IsSorted}[\langle\langle L, m, R\rangle, n, \epsilon\rangle]$

P-60. $\mathit{IsSorted}[\langle L, m, R\rangle] \wedge n \leq \mathit{LfM}[\langle L, m, R\rangle] \Longrightarrow \mathit{IsSorted}[\langle \epsilon, n, \langle L, m, R\rangle\rangle]$

P-61. $\mathit{RgM}[\langle L_1, n, R\rangle] = \mathit{RgM}[\langle L_2, n, R\rangle]$

P-62. $\mathit{LfM}[\langle L, n, R_1\rangle] = \mathit{LfM}[\langle L, n, R_2\rangle]$

P-63. $\mathit{Insert}[n, T] \neq \epsilon$

P-64. $\mathit{LfM}[\langle \mathit{Insert}[n, T], m, T_1\rangle] = n \vee \mathit{LfM}[\langle \mathit{Insert}[n, T], m, T_1\rangle] = \mathit{LfM}[\langle T, m, T_1\rangle]$

P-65. $\mathit{RgM}[T_1, m, \langle \mathit{Insert}[n, T]\rangle] = n \vee \mathit{RgM}[\langle T_1, m, \mathit{Insert}[n, T]\rangle] = \mathit{RgM}[\langle T_1, m, T\rangle]$

P-66. $F_1[\langle L, n, R\rangle] \neq \epsilon$

All these properties are mechanically proved, and based on them also the theorem stating that F_1 is a sorting algorithm:

Theorem 1. $\underset{T}{\forall}\Big(\mathit{IsSorted}[F_1[T]]\Big)$

VI. Conclusions

Verification of the sorting algorithms needs a correct formalization of the domain of binary trees, which also contains all the necessary properties needed for the correctness proofs. The construction of such a correct and sufficient formalization has been shown possible, and moreover it can support the actual synthesis of various sorting algorithms. The Coq certification of the synthesized algorithms can be seen as unit tests for checking the soundness of the presented approach, the generated theory and the implemented inference rules in *Theorema*. Moreover, this case study on theory exploration demonstrates that the formalization is not a trivial task and it yields numerous notions and properties which are necessary in the process of algorithm synthesis by proving, and surely also in the process of certification.

Using the presented approach one manages to build a knowledge base which is useful in concrete applications of binary trees – in this case for the automatic proof-based synthesis of algorithms on binary trees.

Not all the presented properties have been certified by Coq. In the future, it would be interesting to build a Coq library with the proposed theory of binary trees and, on the other hand, certify the synthesized sorting algorithms other than *Sort-1*.

Acknowledgements

Isabela Drămnesc: This work was partially supported by the strategic grant POSDRU/159/1.5/S/137750, Project Doctoral and Postdoctoral programs support for increased competitiveness in Exact Sciences research cofinanced by the European Social Fund within the Sectoral Operational Programme Human Resources Development 2007 – 2013.

References

[1] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[2] A. Bouhoula and F. Jacquemard. Automated Induction for Complex Data Structures. *CoRR*, abs/0811.4720, 2008.

[3] B. Buchberger. Theory Exploration with Theorema. In *Analele Universitatii Din Timisoara, Ser. Matematica-Informatica*, volume XXXVIII, pages 9–32, 2000.

[4] B. Buchberger. Algorithm Supported Mathematical Theory Exploration: A Personal View and Strategy. In *Proceedings of AISC'04*, volume Springer, pages 236–250, 2004.

[5] B. Buchberger et al. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 4(4):470–504, 2006.

[6] A. Craciun and M. Hodorog. Decompositions of Natural Numbers: From A Case Study in Mathematical Theory Exploration. In *Proceedings of SYNASC'07*, 2007.

[7] I. Dramnesc and T. Jebelean. Proof Techniques for Synthesis of Sorting Algorithms. In *Proceedings of SYNASC'11*, pages 101–109. IEEE Computer Society, 2011.

[8] I. Dramnesc and T. Jebelean. Automated Synthesis of Some Algorithms on Finite Sets. In *Proceedings of SYNASC'12*, pages 143–151. IEEE Computer Society, 2012.

[9] I. Dramnesc and T. Jebelean. Theory Exploration in Theorema: Case Study on Lists. In *Proceedings of SACI'12*, pages 421–426. IEEE Xplore, 2012.

[10] I. Dramnesc and T. Jebelean. Theory Exploration of Sets Represented as Monotone Lists. In *Proceedings of SISY'14*. IEEE Xplore, 2014.

[11] I. Dramnesc and T. Jebelean. A Case Study in Proof Based Synthesis of Algorithms on Monotone Lists. In *Proceedings of SACI'15*, pages 483 – 488. IEEE Xplore, 2015.

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

[13] 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.

[14] D. E. Knuth. *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*. Addison-Wesley, 1998.

[15] K. Kunen. *Set Theory: An Introduction to Independence Proofs*. Studies in Logic and the Foundations of Mathematics. Elsevier, 1980.

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

[17] S. Stratulat. A Unified View of Induction Reasoning for First-Order Logic. In A. Voronkov, editor, *Turing-100 (The Alan Turing Centenary Conference)*, volume 10 of *EPiC Series*, pages 326–352. EasyChair, 2012.

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