Abstract—We present the systematic build-up in the Theorema system of the theory of lists. This was carried out in parallel with the process of synthesis of some sorting algorithms in the same system. We use appropriate induction principles for lists and we construct a collection of properties of lists which are necessary for the automatic synthesis of sorting algorithms.

In contrast with another version of the list theory in the Theory system, which is based on higher order logic and uses sequence variables, our approach uses first order predicate logic (which is semi-decidable). This approach opens the way for the effective automation of proofs, of the exploration of theories and of the synthesis of the algorithms applied on lists. This case study in theory exploration can be also used in teaching, especially because it is completely supported by the Theorema system.

Index Terms—theory; exploration; lists; Theorema;

I. INTRODUCTION

In our previous work, during the sorting algorithm synthesis process, see [4], [5], we use certain definitions and properties of some notions. In this paper we show how we add these notions and their corresponding properties, we check all the new notions introduced and we prove all the propositions using the new prover implemented by us in the Theory system.

The implementation of the new prover and also the case studies that we present in this paper are carried out in the frame of the Theory system (www.theorema.org and e.g. [2]) which is implemented in Mathematica – see [9]. We use this system because it offers support for theory exploration, see [1], and the proofs are generated in natural style. Full details of the case study that we present in this paper are given in [6].

Applications. The immediate purpose of the theory developed here is the automated synthesis of sorting algorithms. However, other applications are obviously possible, for instance reasoning about and operating on sets represented as sorted lists without repetitions, and even more importantly reasoning about lists in the context of program verification.

Related Work: An early partial formalization of the theory of lists is presented in [8], however not in natural style, and not related to a complex application like automated synthesis.

Bruno Buchberger introduces a scheme-based model for theory exploration in the Theory system, see [3], which is applied for natural numbers in [7]. We explore the theory of lists which is carried out in parallel with the process of synthesis of some sorting algorithms.

II. REASONING

We implemented a prover in the Theory system which is based on inference rules like:

- Rewriting rules like rewrite by definitions.
- Special inference rules which result from lifting some properties from the knowledge base to the inference level.
- Matching and unification. The most general case is unifying a conjunctive goal with an universal assumed implication. For instance if the assumption is (FormLHS ⇔ FormRHS) and the goal is Goal1 ∧ Goal2 it first tries to do matching on Goal1 with FormRHS (or on Goal2 with FormRHS) of the assumed implication and if the matching is done we obtain a substitution, then it is sufficient to prove FormLHS ∧ Goal2 (or FormLHS ∧ Goal1) (under the appropriate substitution).

Induction is lifted to inference.

Head-Tail-Induction1:

\[
\left( P(\langle \rangle) \land \forall a, X (P[X] \implies P[a \sim X]) \right) \implies \forall X P[X]
\]

Head-Tail-Induction2:

\[
\forall a \left( P[a \sim \langle \rangle] \land \forall X (P[X] \implies P[a \sim X]) \right) \implies \forall X P[X]
\]

III. THE EXPLORATION OF LIST THEORY

In this section we describe in general the method that we use for the exploration of the theory of lists.

We build the theory of lists in a bottom-up way. In order to construct the lists we use a principle based on two constructors, namely: \(\sim\) and \(\langle \rangle\). We use the notations: \(\langle \rangle\) for the empty list, \(a \sim \langle \rangle\) for a list containing one single element \(a\) and \(a \sim A\) is the list having the first element \(a\) and the rest of the list \(A\). We use these notations because we want to use the first order predicate logic (as much as we can) which is semi-decidable and this approach increases the feasibility of proving.

We start by adding new notions.

A. Basic notions

1) The function HeadOf:

Definition 1. \(\forall a, X (\text{HeadOf}[a \sim X] = a)\)
2) The function TailOf:

Definition 2. \( \forall_{a,X} (\text{TailOf}[a \sim X] = X) \)

3) The function LastElem:

Definition 3. \( \forall_{a,X} (\text{LastElem}[a \sim \{\}] = a \quad a \neq \{\}) \)
\( \text{LastElem}[a \sim \{\}] = \text{LastElem}[X] \)

4) “\(\sim\)” Adding at the end:

Definition 4. \( \forall_{a,b,X} (\{ \sim a = a \sim \{ \}) \)
\( a \neq \{ \quad (a \sim b) \implies (a \sim (b \sim X)) \quad (a \sim X) \sim Y = a \sim (X \sim Y) \)

We add the property:

Proposition 1 (“Commutativity-1”), \( \forall_{X} (X \sim \{\} = X) \)

This property is proved very easy by applying the Head-Tail-Induction\(1\) on \(X\) and by the Definition 5.

6) The predicate “\(<\)” – Occurs in:

Definition 6. \( \forall_{a,b,X} (a \neq \{ \quad a \prec (a \sim X) \quad (a \neq b) \implies (a \prec (b \sim X) \iff (a \prec X)) \) \)

We add the property:

Proposition 2. \( \forall_{X} (a \prec (a \sim \{\})) \)

This property is proved very easy by the Definition 6.

7) The function “dfo” – Deletes the first ocurrence of an element from a list:

Definition 7. \( \forall_{a,b,X} (dfo[a, \{\}] = \{ \quad dfo[a, a \sim X] = X \quad (a \neq b) \implies (dfo[a, b \sim X] = b \sim dfo[a, X]) \) \)

8) The predicate “\(\approx\)” – Have the same elements:

Definition 8. \( \forall_{a,X,Y} ((a \prec Y) \land X \approx dfo[a, Y]) \implies ((a \prec X) \approx Y) \)

We add the following properties:

Proposition 3 (“Reflexivity”), \( \forall_{X} (X \approx X) \)

We prove this property by applying the Head-Tail-Induction\(1\) on \(X\) and by the Definition 8, the Definition 6 and by the Definition 7.

Proposition 4 (“Symmetry”), \( \forall_{X,Y} (X \approx Y \iff Y \approx X) \)

The Proposition 4 is lifted to the inference level an it is used for generating local ground assumptions. A local assumption is an assumption available only for that particular case of proving and a ground assumption is an assumption which do not contain variables or meta-variables. For e.g. if we have a local assumption \(A_0 \approx B_0\), then we generate also the assumption \(B_0 \approx A_0\). In this way we obtain more efficient proofs.

Proposition 5 (“Transitivity”), \( \forall_{X,Y,Z} (X \approx Y \land Y \approx Z \implies X \approx Z) \)

These two properties are proved by applying the Head-Tail-Induction\(1\) on \(X\), on \(Y\), respectively on \(Z\) and by the Proposition 3, by the Definition 8, the Definition 6 and by the Definition 7.

Proposition 6.
\( \forall_{a,X,Y,Z} ((X \approx a \sim Y \land X \approx Z) \implies (X \approx a \sim Z)) \)

The Proposition 6 is proved very easy by the Proposition 5 and we need to add in the knowledge base the Proposition 11 (see below).

Proposition 7.
\( \forall_{a,b,X,Y} ((a \sim X \approx Y \land X \approx Z) \implies (a \sim Z \approx Y)) \)

We prove the Proposition 7 by applying the Proposition 4 on the local assumptions and by the Proposition 5.

Proposition 8.
\( \forall_{a,b,X,Y} ((X \approx a \sim Y) \implies (b \sim X \approx a \sim (b \sim Y))) \)

We prove the Proposition 8 by applying the Proposition 4 on the local assumptions, by the Proposition 7 and we also need to add in the knowledge base the Proposition 13 (see below).

Proposition 9.
\( \forall_{a,b,X,Y} ((b \sim X \approx Y) \implies (b \sim (a \sim X) \approx a \sim Y)) \)

We prove the Proposition 9 by the Proposition 6 and by the Proposition 13.

Proposition 10.
\( \forall_{a,X,Y,Z} ((X \approx Y \approx Z) \implies (a \sim X \approx Y \approx a \sim Z)) \)

We prove the Proposition 10 by applying the Proposition 4 on the local assumption, by the Proposition 7 and we also need to add in the knowledge base the Proposition 14 (see below).

Proposition 11.
\( \forall_{a,X,Y} ((X \approx Y) \implies (a \sim X \approx a \sim Y)) \)

This property is proved by the Proposition 4, by the Proposition 7 and by the Proposition 3.

Proposition 12.
\( \forall_{a,X,Y} ((X \approx Y) \implies (a \sim X \approx Y \approx a)) \)
We prove the Proposition 12 by using the Proposition 7, the Proposition 4 and we also need to add in the knowledge base the Proposition 15 (see below).

The properties from Proposition 8 to Proposition 12 are lifted to the inference level. The inference rule is based on the fact that if an element appears on the left hand side and also on the right hand side of the equivalence, then it can be deleted.

**Proposition 13.**
\[ \forall_{a,b,X} \left( b \sim (a \sim X) \equiv a \sim (b \sim X) \right) \]

The Proposition 13 is proved by the Definition 8, by Definition 7, by the Proposition 11 and by Proposition 3.

**Proposition 14.**
\[ \forall_{a,X,Y} \left( a \sim X \equiv Y \equiv Y \sim a \right) \]

For proving this proposition we need to add in the knowledge base the Proposition 16 (see below).

**Proposition 15.**
\[ \forall_{a,X} \left( a \sim X \equiv X \sim a \right) \]

We prove the Proposition 15 by Head-Tail-Induction1 on X, by using the Definition 4, the Proposition 3 and the Proposition 9.

**Proposition 16.**
\[ \forall_{X,Y} \left( X \equiv Y \equiv Y \approx X \right) \]

**Proposition 17.**
\[ \forall_{X,Y,Z} \left( (X \sim Z \equiv Z \sim U \sim Y) \implies (X \sim Y \approx Z) \right) \]

**Proposition 18.**
\[ \forall_{X,Y,Z} \left( (X \sim Y \approx Z \sim V \approx Z) \implies (X \sim Y \approx Z) \right) \]

We prove these properties by applying the Head-Tail-Induction1 on X, on Y, respectively on Z, U, V and by using the definitions and the propositions introduced by now in the knowledge base.

The Proposition 6, Proposition 7, Proposition 17 and Proposition 18 are lifted to the inference level. The inference rule is based on the fact that if an element from an expression is equivalent with something, then replace in expression the equivalent part.

**Proposition 19.**
\[ \forall_{X \neq \emptyset} \left( X \approx \text{HeadOf}[X] \sim \text{TailOf}[X] \right) \]

This property is proved by applying the Head-Tail-Induction2 on X and by the Definition 1, the Definition 2 and by the Proposition 3.

9) The predicates “IsSorted” and “IsSortedDec”:

**Definition 9.**
\[ \forall_{a,b,X} \left( \begin{array}{l}
\text{IsSorted}([\emptyset]) \\
\text{IsSorted}([a \sim \emptyset]) \\
\text{IsSorted}([b \sim X] \land (a \leq b)) \implies \\
\text{IsSorted}([a \sim (b \sim X)])
\end{array} \right) \]

Similarly, we have the definition of “IsSortedDec”, but we change \( b \leq a \).

**B. Ordering**

This subsection contains the additional properties necessary for ordered lists. First we list the definitions and the properties for ascending ordered lists, then we discuss the necessary modifications for the descending ordered lists. Ordered lists are very important because they occur in many practical applications (as for instance sorting) and also because they are often used for the representations of sets.

Because we do not use explicitly the types of the objects we use predicate and function symbols which cannot be overloaded. We use the total non-strict ordering \( \leq \) to denote the order between elements. \( a \leq b \) denotes that the element \( a \) is smaller or equal to the element \( b \). \( a \preceq A \) denotes that the element \( a \) is smaller than all the elements from the list \( A \); \( A \preceq a \) denotes that all the elements from the list \( A \) are smaller than the element \( a \); \( A \ll B \) denotes that all the elements from the list \( A \) are smaller than all the elements from the list \( B \).

**Definition 10 (“\( \preceq \)”):**
\[ \forall_{a,b,X} \left( \begin{array}{l}
a \preceq \emptyset \\
(a \leq b \land a \preceq X) \implies (a \preceq b \sim X)
\end{array} \right) \]

**Definition 11 (“\( \ll \)”):**
\[ \forall_{a,b,X} \left( \begin{array}{l}
\emptyset \ll a \\
(b \leq a \land X \ll a) \implies (b \sim X \ll a)
\end{array} \right) \]

**Definition 12 (“\( \ll \)”):**
\[ \forall_{a,b,X,Y} \left( \begin{array}{l}
a \ll Y \land X \ll Y \implies (a \sim X \ll Y) \\
(X \ll b \land X \ll Y) \implies (X \ll b \sim Y)
\end{array} \right) \]

Properties:

**Proposition 20.** \( \forall_{a} (a \leq a) \)

**Proposition 21.** \( \forall_{a} (a \preceq a \sim \emptyset) \)

**Proposition 22.** \( \forall_{a} (a \sim \emptyset \ll a) \)

**Proposition 23.** \( \forall_{a} (\emptyset \ll a \sim \emptyset) \)

**Proposition 24.** \( \forall_{a} (a \sim \emptyset \ll \emptyset) \)

These properties are proved very easy by the definitions above and by Proposition 20.

**Proposition 25.** \( \forall_{b,X,Y} \left( (X \approx Y \land b \preceq X) \implies (b \approx Y) \right) \)

**Proposition 26.**
\[ \forall_{a,b,Y} \left( (a \leq b \land b \preceq Y) \implies a \preceq b \sim Y \right) \]

**Proposition 27.**
\[ \forall_{a,b,X,Y} \left( (a \preceq X \land X \ll Y) \implies a \sim X \ll Y \right) \]
The Propositions 26 and 27 are lifted to the inference level. The inference rule is based on rewriting the right hand side into the left hand side of the implication.

**Proposition 28.**
\[ \forall_{a, X} \left( (a \preceq X \land \text{IsSorted}(X)) \implies \text{IsSorted}(a \prec X) \right) \]

**Proposition 29.**
\[ \forall_{a, b, X} \left( (a \preceq b \land b \preceq X \land \text{IsSorted}(X)) \implies \text{IsSorted}(a \prec (b \prec X)) \right) \]

**Proposition 30.**
\[ \forall_{X, Y} \left( \text{IsSorted}(X) \land X \ll Y \land \text{IsSorted}(Y) \implies \text{IsSorted}(X \ll Y) \right) \]

**Proposition 31.**
\[ \forall_{a, X, Y} \left( (a \preceq X \land \text{IsSorted}(X) \land X \ll Y \land \text{IsSorted}(Y)) \implies \text{IsSorted}(a \prec (X \ll Y)) \right) \]

**Proposition 32.**
\[ \forall_{a, X, Y} \left( \text{IsSorted}(X) \land X \ll Y \land \text{IsSorted}(Y) \land Y \preceq a \implies \text{IsSorted}(X \ll Y \prec a) \right) \]

**Proposition 33.**
\[ \forall_{a, X, Y} \left( \text{IsSorted}(X) \land X \ll a \land a \preceq Y \land \text{IsSorted}(Y) \implies \text{IsSorted}(X \ll a \prec Y) \right) \]

Similarly, we have the properties for “IsSortedDec”. All these properties are proved by applying the Head-Tail-Induction1 and by using the definitions and the properties that we already have in the knowledge base.

All the properties from 28 to 33 are lifted to inference. We choose to lift these properties because it is always safe to rewrite a formula of the form \( \text{IsSorted}(\text{expr}) \) into the left hand side of the implication (from properties depending on the situation). By lifting these properties to the inference we gain time in proving because we eliminate the alternative branches and no backtracking is needed.

C. Some Sorting Algorithms and their Corresponding Auxiliary Functions

1) **Selection-Sort**: consists in repeatedly selecting the minimal element from the list and moving it at the beginning. This algorithm needs functions for selecting and removing the minimal element, which in turn need a function for finding the minimal of two elements:

   The function “\( \text{min2} \)”:

   \[ \forall_{a, b} \left( (a \leq b) \implies \text{min2}(a, b) = a \right) \]

   \[ \forall_{a, b} \left( \neg(a \leq b) \implies \text{min2}(a, b) = b \right) \]

The following two properties of this function are necessary for the further build-up of the theory:

**Definition 13.**
\[ \forall_{a, b} \left( (a \leq b) \implies \text{min2}(a, b) = a \right) \]

**Proposition 34.**
\[ \forall_{a, y, X \neq \emptyset} \left( y \preceq X \implies \text{min2}(a, y) \prec a \prec X \right) \]

**Proposition 35.**
\[ \forall_{a, y, X \neq \emptyset} \left( y \preceq X \implies \text{min2}(a, y) \preceq a \prec X \right) \]

The function “\( \text{MinElem} \)”:

**Definition 14.**
\[ \text{MinElem}(a \prec \emptyset) = a \]

\[ \forall_{a, b, X} \left( \text{MinElem}(a \prec (b \prec X)) = \text{min2}(a, \text{MinElem}(b \prec X)) \right) \]

The function “\( \text{delME} \)”:

**Definition 15.**
\[ \forall_{X \neq \emptyset} \left( \text{delME}(X) = \text{dfo}(\text{MinElem}(X), X) \right) \]

These two properties are particularly useful:

**Proposition 36.**
\[ \forall_{X \neq \emptyset} \left( X \approx (\text{MinElem}(X) \prec \text{delME}(X)) \right) \]

**Proposition 37.**
\[ \forall_{X \neq \emptyset} \left( \text{MinElem}(X) \preceq \text{delME}(X) \right) \]

We prove these two properties by Head-Tail-Induction2 on \( X \) and by the Definition 14, the Definition 7, the Definition 10, the Definition 15, the Definition 13, the Definition 8, the Proposition 25 and by the Proposition 26.

The **Selection-Sort** algorithm:

The algorithm is defined by the following formulae:

**Algorithm 1.**
\[ \text{SelSort}(\emptyset) = \emptyset \]

\[ \forall_{a, X} \left( \text{SelSort}(a \prec X) = \text{MinElem}(a \prec X) \prec \text{SelSort}(\text{delME}(a \prec X)) \right) \]

The following properties assert that this function indeed sorts a list (that is, it transforms its argument in a sorted list having the same elements).

**Proposition 38.**
\[ \forall_{X} \left( \text{IsSorted}(\text{SelSort}(X)) \right) \]

**Proposition 39.**
\[ \forall_{X} \left( X \approx \text{SelSort}(X) \right) \]

2) **Insertion-Sort**: consists in recursively splitting the current list into the head and the tail of the list, sort the tail and insert the head into the sorted list such that the result will be a sorted list. This algorithm needs the following function:

The function “Insertion”:

**Definition 16.**
\[ \forall_{a, X} \left( \text{Insertion}(a, X) = a \prec X \right) \]

\[ \forall_{a, b, X} \left( \text{Insertion}(a, b \prec X) = a \prec (b \prec X), \text{ if } a \leq b \right) \]

\[ \forall_{a, b, X} \left( \text{Insertion}(a, b \prec X) = b \prec \text{Insertion}(a, X), \text{ if } \neg(a \leq b) \right) \]

The following properties are particularly useful:
Proposition 40. 
\[ \forall_{a,X} (a \vdash Insertion[a, X]) \]

Proposition 41. 
\[ \forall_{a,X,Y} (X \approx Y) \implies (X \approx dfo[a, Insertion[a, Y]]) \]

Proposition 42. 
\[ \forall_{a,X} (X \approx dfo[a, Insertion[a, X]]) \]

Proposition 43. 
\[ \forall_{a,X} (IsSorted[X] \implies IsSorted[Insertion[a, X]]) \]

Proposition 44. 
\[ \forall_{a,X} (a \vdash X \approx Insertion[a, X]) \]

These properties are proved by applying the Head-Tail-Induction and by using the definitions and the properties that we already have in the knowledge base.

The “Insertion-Sort” algorithm:

Algorithm 2.
\[ \forall_{a,X} \left( \text{InsSort}([], []) = \langle \rangle \right) \]
\[ \forall_{a,X} \left( \text{InsSort}[a \vdash X] = \text{Insertion}[a, \text{InsSort}[X]] \right) \]

The properties of this algorithm are similar with the ones from the Selection-Sort algorithm (the Proposition 38 and the Proposition 39).

3) Quick-Sort: consists in recursively splitting the current list into two sublists, sort the two sublists and then concatenate them and obtain a sorted list. This algorithm needs the following functions:

The functions “SelectLess” and “SelectSmall”:

Definition 17.
\[ \forall_{a,b,X} \left( \text{SelectLess}[a, b \vdash X] = b \vdash \text{SelectLess}[a, X], \text{if } b \leq a \right) \]
\[ \forall_{a,b,X} \left( \text{SelectLess}[a, b \vdash X] = \text{SelectLess}[a, X], \text{if } \neg(b \leq a) \right) \]

Definition 18.
\[ \forall_{X \neq \langle \rangle} \left( \text{SelectSmall}[X] = \text{SelectLess}[\text{HeadOf}[X], \text{TailOf}[X]] \right) \]

The functions “SelectGreater” and “SelectBig”:

Definition 19.
\[ \forall_{a,b,X} \left( \text{SelectGreater}[a, b \vdash X] = \text{SelectGreater}[a, X], \text{if } b \leq a \right) \]
\[ \forall_{a,b,X} \left( \text{SelectGreater}[a, b \vdash X] = b \vdash \text{SelectGreater}[a, X], \text{if } \neg(b \leq a) \right) \]

Definition 20.
\[ \forall_{X \neq \langle \rangle} \left( \text{SelectBig}[X] = \text{SelectGreater}[\text{HeadOf}[X], \text{TailOf}[X]] \right) \]

Proposition 45.
\[ \forall_{X \neq \langle \rangle} \left( X \vdash \text{SelectSmall}[X] \vdash \text{SelectBig}[X] \right) \]

Proposition 46.
\[ \forall_{X \neq \langle \rangle} \left( \text{SelectSmall}[X] \vdash \text{SelectBig}[X] \right) \]

The “Quick-Sort” algorithm:

Algorithm 3.
\[ \forall_{a,X} \left( \text{QSort}([\langle \rangle]) = [\langle \rangle] \right) \]
\[ \forall_{a,X} \left( \text{QSort}[a \vdash \langle \rangle] = a \vdash \langle \rangle \right) \]
\[ \forall_{a,X} \left( \text{QSort}[X] = \text{QSort}[\text{SelectSmall}[X]] \vdash \text{QSort}[\text{SelectBig}[X]] \right) \]

The properties of this algorithm are similar with the ones from the other sorting algorithms previously introduced.

4) Merge-Sort: consists in recursively splitting the current list into two sublists, sort the two sublists and then combine (merge) them such that the result will be a sorted list.

The function “OddElem”: This function returns the list of elements on the odd positions from a list.

Definition 21.
\[ \forall_{a,b,X} \left( \text{OddElem}[a \vdash X] = a \vdash \text{OddElem}[X] \right) \]

The function “EvenElem”: This function returns the list of elements on the even positions from a list.

Definition 22.
\[ \forall_{a,b,X} \left( \text{EvenElem}[a \vdash X] = b \vdash \text{EvenElem}[X] \right) \]

Proposition 47.
\[ \forall_{X \neq \langle \rangle} \left( X \approx \text{OddElem}[X] \vdash \text{EvenElem}[X] \right) \]

The function “Merge”:

Definition 23.
\[ \forall_{a,b,X,Y} \left( \text{Merge}[a \vdash X, b \vdash Y] = \begin{cases} a \vdash \text{Merge}[X, b \vdash Y], \text{if } a \leq b \\ b \vdash \text{Merge}[a \vdash X, Y], \text{if } \neg(a \leq b) \end{cases} \right) \]

The “Merge-Sort” algorithm: is defined by:

Algorithm 4.
\[ \forall_{a,X} \left( \text{MSort}([\langle \rangle]) = [\langle \rangle] \right) \]
\[ \forall_{a,X} \left( \text{MSort}[a \vdash \langle \rangle] = a \vdash \langle \rangle \right) \]
\[ \forall_{a,X} \left( \text{MSort}[X] = \text{Merge}[\text{MSort}[\text{OddElem}[X]], \text{MSort}[\text{EvenElem}[X]]] \right) \]

The properties of this algorithm are similar with the ones from the other sorting algorithms previously introduced.

5) Special Merge-Sort: This algorithm, which we could not find in the literature, came up during our experiments of automatic synthesis. It consists in recursively splitting the current list into a sorted list and an unsorted one, and then merging them like in the Merge-Sort algorithm. (The usefulness and the time complexity of this novel algorithm are currently under investigation.)
The functions “LeftXS” and “LeftXSl”:

**Definition 24.**

\[
\begin{align*}
\text{LeftXS}([], U) &= U \\
\text{LeftXS}(a, [], U) &= \text{LeftXS}(b, X, a, [], U) \\
\forall_{a, X, U} \left( \text{LeftXS}(a, X, a, U) = \text{LeftXS}(b, X, a, U, R) \right)
\end{align*}
\]

The properties of this algorithm are similar with the ones from the other sorting algorithms previously introduced.

**Proposition 48.**

\[
\forall_X \left( \text{isSorted}(\text{LeftXS}[X]) \right)
\]

The functions “RestX” and “RestXr”:

**Definition 25.**

\[
\begin{align*}
\forall_{a, X, U} \left( \text{RestX}(a, X, U) = \text{RestX}(a, X, U, R) \right)
\end{align*}
\]

**Proposition 49.**

\[
\forall_X \left( \text{RestXr}[X] = \text{RestX}[X, \emptyset, \emptyset] \right)
\]

The “Special Merge-Sort” algorithm:

**Algorithm 5.**

\[
\begin{align*}
\forall_{a, X} \left( \text{SpMSort}(\emptyset) &= \emptyset \\
\text{SpMSort}(a, X) &= a, \emptyset \\
\text{SpMSort}[X] &= \text{Merge}([\text{LeftXS}[X]], \text{SpMSort}[\text{RestXr}[X]])
\end{align*}
\]

The properties of this algorithm are similar with the ones from the other sorting algorithms previously introduced.

**IV. CONCLUSION AND FURTHER WORK**

Classical simple domains like lists (also known as tuples) are so intuitive and well known that most programmers do not even feel the need of a systematic formalization of their properties. However, lack of systematic treatment may seldom, but unexpected, lead to subtle programming errors, which are often very difficult to track\(^1\). Moreover, the automation of the verification process requires a complete and absolutely correct formalization of the domains involved in the applications which need to be verified. As we show above, such a formalization is not a trivial task, as it yields often surprising (although obvious) properties which are necessary in the process of reasoning about algorithms.

Our work demonstrates that the process of finding relevant notions, definitions, and properties can be significantly boosted by embedding it in a concrete application – in our case the automatic synthesis of sorting algorithms based on proofs of existence of the sorted solution. By identifying the appropriate notions and properties necessary in the process of proving, as well as the additional simpler notions and properties necessary for proving these properties, we have been able to construct a knowledge base which will be certainly useful for the investigation of other applications of lists.

Furthermore the process of theory exploration, which was performed here “by hand” constitutes a useful case study for the investigation of possible methods for the automation of the theory exploration.

A natural extension of this work is the investigation of further algorithms on lists – for instance the ones occurring in the theory of finite sets represented as ordered lists without repetitions.

**REFERENCES**


\(^1\) Unfortunately without concrete references because of confidentiality agreements, we can mention a practical situation where the software department of a major international telephone company fought for three years against a little software bug due to the confusion between addressing an array starting from 0 or from 1. The bug was finally discovered by an academic team using standard techniques from model checking.