A Case Study in Proof Based Synthesis of Algorithms on Monotone Lists

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

Abstract—We apply the synthesis method introduced in our previous work in order to synthesize from proofs certain algorithms operating on sorted lists without duplications (“monotone lists”). The corresponding prover and algorithm extractor are implemented in the Theorema system. Three algorithms are automatically discovered from proofs: symmetrical difference, cartesian product and the cardinality.

The larger context of this work is the manipulation of sets represented as monotone lists. This is a case study in which we demonstrate how to generate automatically the necessary functions, starting from properties from set theory. The properties of sets and of monotone lists which are necessary for the proofs are collected systematically in a knowledge base which extends the one presented in our previous work. This process of theory exploration is supported by a special prover which is able to prove automatically all the statements which are logical consequences of the axioms.

Keywords—automated reasoning, algorithm synthesis, theory exploration, Theorema

I. INTRODUCTION

Sets are used in many algorithms, therefore an efficient implementation of operations with sets is crucial. The context of the research presented here is a larger case study in which we investigate the possibility of automatic synthesis of functions operating on sets represented as monotone lists. The synthesis method is proof based: for the synthesis of a certain function, we start from its specification and we generate automatically the proof of the statement. The proof techniques insure that the proof is constructive, such that the algorithm can be easily extracted from it. For this approach to work, one also needs to “explore” the theory of sets and the theory of monotone lists, that is to add in a systematic manner the necessary axioms and properties which are necessary in the proofs. (The properties are also automatically proved.)

In [3] we demonstrate how to represent sets as monotone lists. For the representation, we introduce two functions $R$ (from sets to their representations) and $S$ (the inverse of $R$) – see Section III. The synthesis process consists in: building an inductive proof, introduce and apply an arsenal of strategies and inference rules which are specific to the domain of sets and of monotone lists, and from the proof extract the corresponding algorithm. During the synthesis process we add to the knowledge base the necessary properties and in this way we explore the theory of sets and of monotone lists. The exploration theory is presented in [4]. The synthesis method and the theory exploration were carried out in the frame of the Theorema system (http://www.risc.jku.at/research/theorema/software, [2]) which is itself implemented in Mathematica [6]. The proofs are easy to follow because they are generated in natural style, similar to human proving.

One purpose of this paper is to show that by applying the proof based synthesis method from [3] we discover from proofs three more algorithms operating on monotone lists. The prover and the extractor which we describe in [3] are extended for the purpose of the present case study. A second purpose is to show how the theory which was explored in [4] is extended with new properties necessary for the proof-based synthesis of the new algorithms. Also, the prover which was implemented in the Theorema system, in order to generate the proofs of all properties from [4], is extended. More inference rules were implemented such that the prover is able to generate the proofs of all properties presented in this paper.

Our case study extends the previous work performed in the Theorema group on theory exploration [1] and on set theory [5].

II. CONTEXT

For lists and for sets we employ the notations used in [3] and [4]. We use $X,Y$ for sets and $A,B$ for monotone lists. We consider the strict ordering relations between elements of a list (e.g., $a < b$). Also, the type of the objects is not used explicitly, but we use predicate and function symbols which are not overloaded.

Sets: Together with these notations, we also use for sets the following function symbols: $\oplus$ for the symmetrical difference, $\otimes$ for the cartesian product of two sets, $\text{Card}$ which returns the cardinality of a set.

For these notions we have properties in the knowledge base which are presented in Section III.

Monotone lists: Sets are represented as sorted lists without duplication (“monotone lists”). For representation we use two functions $R$ and $S$ which are bijective and reversed to each other. $R$ is the function which applied to a set returns its representation, namely the corresponding monotone list, e.g. $R(\{7,2,5\})$ returns the list $(2 \sim (5 \sim (7 \sim \langle \rangle)))$ (or we use the notation $(2,5,7)$) and $S$ is the function which applied to a list returns its corresponding set, e.g. $S[11 \sim (12 \sim (13 \sim \langle \rangle))]$ returns the set $\{11,12,13\}$.
(14 ⊖ ⟨⟩)) returns the set \{11, 12, 13, 14\}. Attention that \( S \) is always applied to monotone lists only. For monotone lists we introduce the following symbols: the function symbol \( \ominus \) for symmetrical difference, \( \odot \) is the function symbol denoting the cartesian product of two monotone lists, \( \text{Card} \) is the function symbol which returns the cardinality of a monotone list.

A. The Problem

In this context, in order to discover from the proofs more binary algorithms operating on monotone lists, we start to prove the synthesis problem introduced in [3], namely:

**Proposition 1.** \( \forall_{A,B} (S[F[A,B]] = G[S[A], S[B]]) \)

by applying the method introduced in [3]. Where, \( F \) corresponds to the binary algorithm which we discover from the proof and \( G \) is a binary function applied on sets for which we have some properties in the knowledge base.

And in order to discover unary algorithms operating on monotone lists we start to prove the following property:

**Proposition 2.** \( \forall_{A} (S[F[A]] = G[S[A]]) \)

where \( F \) is the algorithm which we want to synthesize by proving and \( G \) is a unary function applied on sets for which we have some properties in the knowledge base.

B. The provers in Theorema

The provers which we implemented in the Theorema system, and which are an extension of the provers from [3] and [4] contain the following types of inference rules: rewriting rules (rewrite by definitions, equality rewriting), special inference rules which result from lifting some properties from the knowledge base to the inference level (and are described in [3]), matching and unification rules, and the following induction principles (presented also in [3]) which are lifted to the inference level:

**Head-Tail Induction:**

\[
(P[\emptyset] \land \forall_{a,A} (P[A] \Rightarrow P[a \cup A])) \Rightarrow \forall_{A} (P[A])
\]

**Head-Tail Double Induction:**

\[
(P[\emptyset], \emptyset) \land \forall_{a,A} (P[A, \emptyset]) \land \forall_{b,B} (P[\emptyset, B]) \land \forall_{a,A,b,B} ((P[\emptyset, B] \land P[a \cup A, \emptyset, B] \land P[A, b \cup B]) \Rightarrow P[a \cup A, b \cup B]) \\
\Rightarrow \forall_{A,B} (P[A,B])
\]

III. Experiments on Monotone Lists

**Common Knowledge base** (which contains 2 definitions and 2 properties from [3]):

**Definition 1.** \( \forall_{a,A} \left( S[\emptyset] = \{a\} \right) \)

This function \( S \) applies on a list and returns its corresponding set.

**Definition 2.** \( \forall_{a,X} \left( R[\emptyset] = \emptyset \right) \)

This function \( R \) applies to a set and returns its corresponding representation (the corresponding monotone list). The function \( \text{Insert} \) (see the definition in [3]) inserts an element on one position in a monotone list such that the result is a monotone list. This function does not allow duplications.

**Proposition 3.** \( \forall_{A,X} \left( S[R[X]] = X \right) \)

**Proposition 4.** \( \forall_{A,B,X,Y} \left( (R[X] = R[Y]) \Rightarrow (X = Y) \right) \)

A. Synthesis of Symmetrical Difference

Proposition 1 becomes:

**Proposition 5.** \( \forall_{A,B} (S[A \ominus B] = S[A] \odot S[B]) \)

We start to prove Proposition 5 in two ways.

1) Version 1: use Definition 3 and the following properties in the knowledge base:

**Definition 3.** \( \forall_{X,Y} (X \odot Y = (X \setminus Y) \cup (Y \setminus X)) \)

The following four properties are from [4]:

**Proposition 6.** \( \forall_{X} (X \cup \emptyset) = X \)

**Proposition 7.** \( \forall_{X,Y} (X \cup Y) = (Y \cup X) \)

**Proposition 8.** \( \forall_{X} (\{\} \cup X) = \{\} \)

**Proposition 9.** \( \forall_{X} ((X \setminus \{\}) = X) \)

And we introduce the following four novel properties in the knowledge base:

**Proposition 10.**

\[
\forall_{a,b,A,B} ((a = b) \Rightarrow (S[a \cup A] \setminus S[b \cup B]) = S[A] \setminus S[B])
\]

**Proposition 11.**

\[
\forall_{a,b,A,B} ((a < b) \Rightarrow (S[a \cup A] \setminus S[b \cup B]) = \{a\} \cup (S[A] \setminus S[B])
\]

**Proposition 12.**

\[
\forall_{a,b,A,B} ((b < a) \Rightarrow (S[a \cup A] \setminus S[b \cup B]) = \{b\} \cup (S[A] \setminus S[B])
\]

**Proposition 13.**

\[
\forall_{a,b,X,Y} ((\{a\} \cup X) \cup (\{a\} \cup Y) = \{a\} \cup (X \cup Y))
\]

**Proof:** Start to prove Proposition 5. By Definition 3, Proposition 5 becomes:

**Proposition 14.**

\[
\forall_{A,B} (S[A \ominus B] = (S[A] \setminus S[B]) \cup (S[B] \setminus S[A]))
\]
Prove Proposition 14 by **Head-Tail Double Induction**.

**Base case 1:** Prove:

\[ S[() \supset \emptyset] = (S[()] \setminus S[()]) \bigcup (S[()] \setminus S[()]) \]  
(1)

Rewrite by Definition 1 and the new goal is:

\[ S[() \supset \emptyset] = (\{\} \setminus \{\}) \bigcup (\{\} \setminus \{\}) \]  
(2)

By Proposition 8 and 9 the goal becomes:

\[ S[() \supset \emptyset] = \{\} \bigcup \{\} \]  
(3)

By Proposition 6 we obtain:

\[ S[() \supset \emptyset] = \{\} \]  
(4)

Apply S7 (described in [3]), namely in order to obtain the solution on this case we apply \( R \) on (4), and based on Proposition 3 and 4 the solution is:

\[ () \supset \emptyset = () \]  
(5)

**Base case 2:** Prove:

\[ S[A \supset \emptyset] = (S[A] \setminus S[()]) \bigcup (S[()] \setminus S[A]) \]  
(6)

Rewrite by Definition 1 and the new goal is:

\[ S[A \supset \emptyset] = (S[A] \setminus \{\}) \bigcup (\{\} \setminus S[A]) \]  
(7)

By Proposition 9 and 8 the goal becomes:

\[ S[A \supset \emptyset] = S[A] \bigcup \{\} \]  
(8)

By Proposition 6 we obtain:

\[ S[A \supset \emptyset] = S[A] \]  
(9)

Similar to the previous case, apply S7 [3] and we obtain the solution:

\[ A \supset \emptyset = A \]  
(10)

**Base case 3:** Prove:

\[ S[(\emptyset \supset \emptyset) \bigcup (S[B] \setminus S[()])] \]  
(11)

Continue the proof similar to the previous case, by using Definition 1, Proposition 8, 9, 7, and 6 and we obtain:

\[ S[(\emptyset \supset \emptyset) \bigcup (S[B] \setminus S[()])] = S[B] \]  
(12)

Apply S7 [3] and we obtain the solution:

\[ (\emptyset \supset \emptyset) \bigcup (S[B] \setminus S[()]) = B \]  
(13)

**Induction step:** Assume:

\[ S[A \supset B] = (S[A] \setminus S[B]) \bigcup (S[B] \setminus S[A]) \]  
(14)

\[ S[(a \sim A) \supset (b \sim B)] = (S[a \sim A] \setminus S[B]) \bigcup (S[B] \setminus S[a \sim A]) \]  
(15)

\[ S[A \equiv (b \sim B)] = (S[A] \setminus S[b \sim B]) \bigcup (S[b \sim B] \setminus S[A]) \]  
(16)

Prove:

\[ S[(a \sim A) \equiv (b \sim B)] = (S[a \sim A] \setminus S[b \sim B]) \bigcup (S[b \sim B] \setminus S[a \sim A]) \]  
(17)

Because goal (17) matches Propositions 10, 11, and 12 we obtain three cases:

**Case 1:** By Propositions 10 the goal (17) becomes:

\[ (S[a \sim A] \equiv (b \sim B)] = (S[A] \setminus S[B]) \bigcup (S[B] \setminus S[A]) \wedge (a = b) \]  
(18)

By (14) we obtain:

\[ S[a \sim A] \equiv (b \sim B)] = S[A \equiv B] \wedge (a = b) \]  
(19)

Apply S7 (from [3]), namely consider that the solution obtained on this branch is:

\[ (a \sim A) \equiv (b \sim B) = A \equiv B \]  
(20)

and continue to prove the remaining goal:

\[ a = b \]  
(21)

Apply S5 (from [3]), namely when we reach a simple goal involving only elements, which cannot be proved or disproved, then this goal becomes the conditional assumption on the corresponding branch. (21) becomes the conditional assumption on this branch.

**Case 2:** By Propositions 11 the goal (17) becomes:

\[ (S[a \sim A] \equiv (b \sim B)] = (S[A] \setminus S[b \sim B]) \bigcup (S[b \sim B] \setminus S[A]) \wedge (a < b) \]  
(22)

By Proposition 13 the new goal is:

\[ (S[a \sim A] \equiv (b \sim B)] = (S[A] \setminus S[b \sim B]) \bigcup (S[b \sim B] \setminus S[A]) \wedge (a < b) \]  
(23)

From (16) we obtain:

\[ S[(a \sim A) \equiv (b \sim B)] = (S[a \sim A] \setminus S[b \sim B]) \bigcup (S[b \sim B] \setminus S[A]) \]  
(24)

Similar to the previous case, apply S7 (from [3]), namely consider that the solution on this branch is:

\[ (a \sim A) \equiv (b \sim B) = a \sim (A \equiv (b \sim B)) \]  
(25)

and continue to prove the remaining goal:

\[ a < b \]  
(26)

Apply S5 (from [3]) and (26) becomes the conditional assumption on this branch.

**Remark:** By using Definition 2, when applying S7 on (24), normally we obtain \( (a \sim A) \equiv (b \sim B) = \text{Inser}(a, A \equiv (b \sim B)) \), but because S applies only to monotone lists, then we can consider the solution to be (25).  

**Case 3:** By Propositions 12 the goal (17) becomes:

\[ (S[(a \sim A) \equiv (b \sim B)] = (((b) \bigcup (S[a \sim A] \setminus S[B])) \bigcup (b \bigcup (S[B] \setminus S[a \sim A])) \wedge (b < a) \]  
(27)
Similar to the previous cases one obtains the solution:

\[(a \sim A) \supseteq B = b \sim ((a \sim A) \supseteq B)\]  \hspace{1cm} (28)

and the conditional assumption on this branch:

\[b < a\]  \hspace{1cm} (29)

In order to extract the algorithm from the proof we apply the strategy S6 [3], namely we check if all the cases are covered (because of the conditional assumptions obtained).

The “\(\supseteq\)” algorithm extracted from the proof: which operates on monotone lists is:

**Algorithm 1.**

\[
\forall_{a,b,A,B} (a \sim A) \supseteq B = B \\
\begin{align*}
(a = b) & \Rightarrow (a \sim A) \supseteq (b \sim B) = \langle⟩, \\
(a < b) & \Rightarrow (a \sim A) \supseteq (b \sim B) = (a \sim (A \supseteq (b \sim B))) \\
(b < a) & \Rightarrow (a \sim A) \supseteq (b \sim B) = (b \sim ((a \sim A) \supseteq B))
\end{align*}
\]

This algorithm is novel.

**Computations:** We make some computations in Theorema with this new algorithm.

- **Input–1:** Compute \([2 \sim (3 \sim (4 \sim (5 \sim (\langle⟩))))] \supseteq (1 \sim (2 \sim ⟨⟩)), \text{ using } \supseteq \rightarrow \text{ Algorithm 1}]

- **Output–1:** \(1 \sim (3 \sim (4 \sim (5 \sim ⟨⟩)))\)

- **Input–2:** Compute \([(1 \sim (2 \sim (3 \sim (4 \sim (5 \sim ⟨⟩))))]) \supseteq (2 \sim (3 \sim (4 \sim (5 \sim ⟨⟩)))), \text{ using } \supseteq \rightarrow \text{ Algorithm 1}]

- **Output–2:** \(1 \sim ⟨⟩\)

2) **Version 2:** use Definition 4 and the following properties in the knowledge base:

**Definition 4.** \(\forall_{X,Y} (X \otimes Y = (X \uplus Y) \setminus (Y \cap X))\)

**Proposition 15.** \(\forall_{a,X,Y} (((a) \cup X) \setminus ((a) \cup Y) = X \setminus Y)\)

We introduce the following six novel properties in the knowledge base:

**Proposition 16.** \(\forall_{a,X,Y} ((a = b) \Longrightarrow S[a \sim A] \cup S[b \sim B] = \{a\} \cup (S[A] \cup S[B]))\)

**Proposition 17.** \(\forall_{a,X,Y} ((a = b) \Longrightarrow S[a \sim A] \cap S[b \sim B] = \{a\} \cup (S[A] \cap S[B]))\)

**Proposition 18.** \(\forall_{a,X,Y} ((a < b) \Longrightarrow S[a \sim A] \cup S[b \sim B] = \{a\} \cup (S[A] \cup S[b \sim B]))\)

**Proposition 19.** \(\forall_{a,X,Y} ((a < b) \Longrightarrow S[a \sim A] \cap S[b \sim B] = \{a\} \cup (S[A] \cap S[b \sim B]))\)

**Proposition 20.** \(\forall_{a,X,Y} ((b < a) \Longrightarrow S[a \sim A] \cup S[b \sim B] = \{b\} \cup (S[a \sim A] \cup S[B]))\)

**Proposition 21.** \(\forall_{a,X,Y} ((b < a) \Longrightarrow S[a \sim A] \cap S[b \sim B] = \{b\} \cup (S[a \sim A] \cap S[B]))\)

**Proof:** Start to prove Proposition 5. By Definition 4, Proposition 5 becomes:

**Proposition 22.** \(\forall_{A,B} (S[A \supseteq B] = (S[A] \cup S[B]) \setminus (S[B] \cap S[A]))\)

The proof of Proposition 22 by **Head-Tail Double Induction** is similar to the one presented above.

From this proof we obtain the same Algorithm 1.

**Remark:** By using 2 different definitions for the symmetrical difference on sets (Definition 3 and 4) and different properties on sets we discover the same algorithm (Algorithm 1).

B. **Synthesis of Cartesian Product**

**Proposition 23.** \(\forall_{A,B} (S[A \supseteq B] = S[A] \otimes S[B])\)

In the knowledge base we add the following properties.

The following three properties are known:

**Proposition 24.** \(\forall_{X} (\{\} \otimes X = \{\})\)

**Proposition 25.** \(\forall_{X} (X \otimes \{\} = \{\})\)

**Proposition 26.** \(\forall_{X,Y} (\neg(X \otimes Y = Y \otimes X))\)

And we introduce the following four properties which are novel:

**Proposition 27.** \(\forall_{a,A,B} (S[a \sim A] \otimes S[B] = \{a\} \cap (S[A] \otimes S[B]))\)

**Proposition 28.** \(\forall_{a,X,Y} (((a) \cup X) \otimes Y = \{a\} \cap (X \otimes Y))\)

**Proposition 29.** \(\forall_{a,b,A,B} \{a\} \otimes \{b\} \cap S[b \sim B] = \{a\} \cup (\{a\} \otimes S[B])\)

**Proposition 30.** \(\forall_{a,b,X} (R[\{a\} \cup X) = \langle(a, b) \rangle, R[X]\})\)

**Proof:** Start to prove Proposition 23 by **Head-Tail Double Induction.**

The three base cases are similar to the ones presented above and we obtain:

**Base case 1:**

\(\langle⟩ \supseteq ⟨⟩ = ⟨⟩\)  \hspace{1cm} (30)
Base case 2:

\[ A_\varnothing(\varnothing) = \varnothing \]  

(31)

Base case 3:

\[ (\varnothing A_\varnothing B) = \varnothing \]  

(32)

Induction step: Assume:

\[ S[(a \sim (\varnothing)) B] = \{a\} \otimes S[B] \]  

(33)

\[ S[A_\varnothing B] = S[A] \otimes S[B] \]  

(34)

\[ S[(a \sim A) B] = S[a \sim A] \otimes S[B] \]  

(35)

\[ S[A_\varnothing(b \sim B)] = S[A] \otimes S[b \sim B] \]  

(36)

And prove:

\[ S[(a \sim A) \varnothing(b \sim B)] = S[a \sim A] \otimes S[b \sim B] \]  

(37)

Rewrite by Definition 1 and the new goal is:

\[ S[(a \sim A) \varnothing(b \sim B)] = \{\{a\} \cup S[A]\} \otimes S[b \sim B] \]  

(38)

By Proposition 28 the goal becomes:

\[ S[(a \sim A) \varnothing(b \sim B)] = \{\{a\} \otimes S[b \sim B]\} \bigcup \{S[A] \otimes S[b \sim B]\} \]  

(39)

From (36) we have to prove:

\[ S[(a \sim A) \varnothing(b \sim B)] = \{\{a\} \otimes S[b \sim B]\} \bigcup S[A_\varnothing(b \sim B)] \]  

(40)

By Proposition 29 the goal becomes:

\[ S[(a \sim A) \varnothing(b \sim B)] = \{\{a, b\} \cup (\{a\} \otimes S[B])\} \bigcup S[A_\varnothing(b \sim B)] \]  

(41)

From (33) we obtain:

\[ S[(a \sim A) \varnothing(b \sim B)] = \{\{a, b\} \cup S[(a \sim (\varnothing)) B]\} \bigcup S[A_\varnothing(b \sim B)] \]  

(42)

Apply S7 [3], by using Proposition 30 and the solution obtained is:

\[ (a \sim A) \varnothing(b \sim B) = \langle (a, b), (a \sim (\varnothing)) B, A_\varnothing(b \sim B) \rangle \]  

(43)

The “\(\varnothing\)” algorithm extracted from the proof: which operates on monotone lists is:

\[
\forall_{a, b, A, B} \begin{cases}
(a \sim A) \varnothing(b \sim B) = \langle (a, b), (a \sim (\varnothing)) B, A_\varnothing(b \sim B) \rangle \\
\end{cases}
\]

This algorithm is novel.

Computation: We make some computations in Theorema with this novel algorithm.

Input-1: Compute([5 \sim (8 \sim (\varnothing))] \varnothing(2 \sim (4 \sim (6 \sim (\varnothing))))), using \(\rightarrow\) Algorithm 2

Output-1: \(\langle 5, 2 \rangle, \langle 5, 4 \rangle, \langle 5, 6 \rangle, \langle 8, 2 \rangle, \langle 8, 4 \rangle, \langle 8, 6 \rangle\)

Input-2: Compute([1 \sim (2 \sim (3 \sim (4 \sim (5 \sim (\varnothing)))))) \varnothing(), using \(\rightarrow\) Algorithm 2

Output-2: \(\varnothing\)

C. Synthesis of Cardinality

Proposition 2 becomes:

**Proposition 31.** \(\forall_A S[\text{Card}[A]] = \text{Card}[S[A]]\)

In the knowledge base we add the following definition:

**Definition 5.** \(\forall_{a, X} \left( \text{Card}[\{a\} \cup X] = 1 + \text{Card}[X] \right)\)

**Proof:** Start to prove Proposition 31 by Head-Tail Induction.

Base case: By Definition 1, and 5 and by applying S7 [3] we obtain the solution:

\[ \text{Card}[\varnothing] = 0 \]  

(44)

Induction step: Assume:

\[ S[\text{Card}[A]] = \text{Card}[S[A]] \]  

(45)

and prove:

\[ S[\text{Card}[a \sim A]] = \text{Card}[S[a \sim A]] \]  

(46)

By Definition 1 the new goal is:

\[ S[\text{Card}[a \sim A]] = \text{Card}[\{a\} \cup S[A]] \]  

(47)

By Definition 5 the new goal is:

\[ S[\text{Card}[a \sim A]] = 1 + \text{Card}[S[A]] \]  

(48)

From (45) we obtain:

\[ S[\text{Card}[a \sim A]] = 1 + \text{Card}[S[A]] \]  

(49)

Apply S7 [3] and the solution obtained is:

\[ \text{Card}[a \sim A] = 1 + \text{Card}[A] \]  

(50)

The “\(\overline{\text{Card}}\)” algorithm extracted from the proof: which operates on monotone lists is:

\[
\forall_{a, A} \begin{cases}
\text{Card}[\varnothing] = 0 \\
\text{Card}[a \sim A] = 1 + \text{Card}[A] \\
\end{cases}
\]

**Remark:** This case study shows how, for unary algorithms, if we have the definition of the function which applies on sets, then for monotone lists we obtain a very similar algorithm.
**Computations:** We make some computations in *Theorema* with this algorithm.

Input–1: Compute $\text{Card}[1 \sqcup (2 \sqcup \langle \rangle)]$, using $\rightarrow$ Algorithm 3

Output–1: 2

Input–2: Compute $\text{Card}[1 \sqcup (2 \sqcup (3 \sqcup (4 \sqcup (5 \sqcup \langle \rangle)))))$, using $\rightarrow$ Algorithm 3

Output–2: 5

IV. CONCLUSIONS

This paper presents the continuation of the case study started in [3].

The result of this work consists in three algorithms operating on monotone lists, which are discovered from proofs by applying the proof–based synthesis introduced in [3], namely, two novel algorithms: the symmetrical difference, the cartesian product, and the cardinality, which is similar to the one operating on sets. Moreover we introduce one more synthesis problem in order to discover unary algorithms. We extend the work from [4] by adding in the knowledge base 5 definitions (2 definitions from [3] and 3 known definitions) and 22 properties for sets and monotone lists (14 properties are novel and 8 properties are known or they are from [3] and [4]). We also develop 2 new provers and an extractor in the *Theorema* system: one prover (synthesizer), which is the extension of the one from [3], one extractor which is the extension of the one from [3], and one prover which is the extension of the prover from [4].

Our work demonstrates that it is possible to generate automatically correct and efficient algorithms starting from function specifications, which is dual to the traditional approach (programming and verification). Moreover we show that by appropriate specific proof strategies and inference rules one can generate efficiently proofs in natural style for the domain of sets and lists.

ACKNOWLEDGMENT

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


