Proof Techniques for Synthesis of Sorting Algorithms

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*—In the context of constructive synthesis of sorting algorithms, starting from the specification of the problem (input and output conditions), the proof of existence of a sorted tuple is performed inductively and we design, implement, and experiment with different proof techniques: First we use a back-chaining mechanism similar to a Prolog engine for first order logic, in which meta-variables are used for finding the existential witnesses. In order to overcome the search space explosion, we introduce various specific prove-solve methods for the theory of tuples. For instance, the equivalence relation on tuples *"have same elements"* can be treated using a normal form based on multisets – this leads to a very efficient inference rule for rewriting. When reasoning about sorting, we also have an ordering relation between elements. We extend this to an ordering between an element and a tuple, and even between tuples. Ordering relations create specific problems in Prolog style reasoning, because of transitivity and reflexivity. We demonstrate that ordering can be treated very efficiently by decomposing atomic statements into simpler ones (containing only symbols instead of terms), both for goals (backward reasoning) as well as for assumptions (forward reasoning). This leads to an interesting combination of backward and forward inferences which goes beyond and complements Prolog style reasoning. Finally, we develop a solving mechanism for finding sorted tuples, which performs the proof more efficiently, by combining relatively simple inference rules and small searches with goal directed solving rules.**

The techniques are implemented in the *Theorema* system and are able to produce automatically proofs and algorithms for various problems: Insertion Sort, Insertion, Merge Sort, and Merge. Besides the special proof techniques, this work also gives useful hints about finding appropriate induction principles for tuples, as well as for the construction of appropriate collections of properties of tuples which are necessary for reasoning about sorting.

***Keywords*-sorting algorithms; synthesis; extraction; proof techniques; induction; *Theorema*;**

I. Introduction

In program synthesis we deal with the following situation: starting from a specification (input and output conditions) of the problem, find an algorithm which satisfies the given specification. For an overview of several methods for synthesizing recursive programs in computational logic see [2]. Our particular approach is similar to the one presented in [8], which details the correctness and the consistency of such inductive synthesis. Very similar problems are also treated in [5], however using sequence variables, which is not a first order logic construct. Our approach is based on first order logic, thus the branches of the inductive proofs are guaranteed to terminate when a corresponding witness exists. In other words, if the respective inductive algorithm exists, then the synthesis process will find it after a finite amount of time.

This finite amount of time could be, however, very long, because of the search performed during the proof. We experiment with various techniques for improving the proof efficiency. Starting from the specification of the sorting problem, the proof of existence of a sorted tuple is performed inductively. The induction may need to be repeated (possibly using different schemes) in order to synthesize some auxiliary functions. This is the "cascading" principle investigated in our previous work, see [10], in which the output specification remains the same, but the input specification grows, thus making the proof easier.

Related to sorting algorithms one needs to deal with other problems too, like e.g. for synthesizing the Insertion Sort algorithm one also needs to synthesize the Insertion algorithm, for synthesizing the Merge Sort algorithm one also needs to synthesize the Merge algorithm.

The implementation of the new provers and also the case studies that we present in this paper are carried out in the frame of the *Theorema* system, therefore we will use similar notations. The *Theorema* system (`www.theorema.org` and e.g. [6]) is implemented in Mathematica – see [18]. We use this system because it offers support for theory exploration and the proofs of the propositions and lemmas are generated in natural style.

Outline: Section II describes the proof based synthesis method which is an extended version of the method presented in [10] – here we use more induction principles. In section III we design the new inference rules in *Prolog style* and also some special inference rules for equivalence and inequalities based on the properties of tuples. The provers based on these techniques are generating the proofs that we present in section IV. From proofs we extract the algorithms: Insertion Sort, Insertion, Merge Sort, and Merge. The process of proving is paralleled with the process of exploring the appropriate theory.

Related Work: Program synthesis in computational logic is a well studied field, you can see a tutorial of program synthesis in [15].

The "lazy thinking" method was introduced in [4] and it is implemented in the *Theorema* system [6], see [5]. In this approach, a "program schemata" (the structure of the solving term) is given to the synthesis procedure. In contrast, in our approach this schemata is also discovered by the procedure, only the inductive principle of the domain is given. Case studies on the synthesis of the Merge-Sort algorithm using schema-based synthesis have been presented in [16], [17]. Different sorting algorithms were developed depending on the details of the proof in [9]. In contrast, our approach better supports the automation of the synthesis process.

In [8] the authors introduce some techniques for constructing induction rules for deductive synthesis proofs. This techniques are based on a combination of "rippling", see [7], and "middle-out reasoning", see [14]. They start from a specification (higher-order existentially quantified variable) and construct a proof. At some proof steps the existential parts of the specification are replaced with meta-variables which will be instantiated during the course of the synthesis proof. This is similar with our approach, where we also use meta-variables which we instantiate during the proof, but, in contrast, they focus on developing induction schemes and we stay in first order logic and we focus on developing proof techniques for tuples. And as soon as the witness is found in the proof we can extract the algorithm from proof, see [1].

In this paper we extend the "cascade" principle that was investigated in [10], see also [11], by introducing and automating new proof techniques for synthesis of sorting algorithms.

All the automatically generated proofs from this paper are presented in detail in [12].

II. Proof Based Synthesis

In this section we describe in general the method that we use for algorithm synthesis.

Given the *problem specification* as two predicates input condition $I_F[X]$ and output condition $O_F[X,Y]$, **find** the *definition* of F such that $\underset{X:I_F}{\forall} O_F[X,F[X]]$. Note that we use square brackets as in $f[x]$ for function application and for predicate application, instead of the usual round parantheses as in $f(x)$. Also, the quantified formula above is a notation for: $(\forall X)(I_F[X] \Longrightarrow O_F[X,F[X]])$. Also by convention, capital letters as A, B, X, Y, Z denote tuples (lists).

We prove $\underset{X:I_F}{\forall}\underset{Y}{\exists} O_F[X,Y]$, by using an induction principle (the choice of the proof style and of the induction principle is not automatic).

A. Induction Principles

We consider that the functions for the composition and the decomposition of tuples come together with the induction principles.

Head-Tail-Induction1:

$$\Big(P[\langle\rangle] \bigwedge \underset{a,X}{\forall}(P[X] \Longrightarrow P[a \backsim X])\Big) \Longrightarrow \underset{X}{\forall} P[X]$$

where $P[X]$ is $I_F[X] \Longrightarrow \underset{Y}{\exists} O[X,Y]$. This corresponds to the usual head-tail inductive definition of tuples.

In our notation, $\langle\rangle$ represents the empty tuple, $a \backsim \langle\rangle$ is a tuple having a single element a, and $a \backsim X$ is the tuple having head a and tail X. Complementary to this inductive definition, we consider that the functions Head and Tail are in the knowledge base. Moreover, in order for this induction scheme to be appropriate for our algorithm, one must have the properties: $I_F[\langle\rangle]$, and $\underset{a}{\forall}\underset{X}{\forall} I_F[a \backsim X] \Longrightarrow I_F[X]$. (That is: the input condition must hold for the base case and for the inductive decomposition of the argument.)

Base case: We prove $\underset{Y}{\exists} O_F[\langle\rangle, Y]$. If the proof succeeds to find a ground term R such that $O_F[\langle\rangle, R]$, then we know that $F[\langle\rangle] = R$.

Induction step: For arbitrary but fixed a and X_0 (satisfying I_F), we assume $\underset{Y}{\exists} O_F[X_0, Y]$ and we prove $\underset{Y}{\exists} O_F[a \backsim X_0, Y]$. We Skolemize the assumption by introducing a new constant Y_0 for the existential Y. The existential variable Y in the goal becomes a metavariable Y^*. If the proof succeeds to find a witness $Y^* = T[a, X_0, Y_0]$ (term depending on a, X_0 and Y_0), then we know that $F[a \backsim X] = T[a, X, F[X]]$.

Algorithm extraction: Finally the algorithm is expressed as:

$$F[X] = \begin{cases} R, & \text{if } X = \langle\rangle \\ T[Head[X], Tail[X], F[Tail[X]]], & \text{if } X \neq \langle\rangle \end{cases}$$

Head-Tail-Induction2: This is the previous induction principle applied to binary predicates:

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

where $P[a,X] : I_F[a \backsim X] \Longrightarrow \underset{Y}{\exists} O[a \backsim X, Y]$.

The proof and the algorithm extraction are similar.

Divide-And-Conquer-Induction:

$$\Big(P[\langle\rangle] \bigwedge \underset{a}{\forall}(P[a \backsim \langle\rangle]) \bigwedge \underset{A,B}{\forall}((P[A] \bigwedge P[B]) \Longrightarrow P[A \asymp B])\Big) \Longrightarrow \underset{A,B}{\forall} P[A \asymp B]$$

where $P[A] : I_F[A] \Longrightarrow \underset{Y}{\exists} O[A,Y]$,
$P[B] : I_F[B] \Longrightarrow \underset{Y}{\exists} O[B,Y]$,
$P[A \asymp B] : (I_F[A] \bigwedge I_F[B]) \Longrightarrow \underset{Y}{\exists} O[A \asymp B, Y]$

We assume that this comes together with two functions *LP: LeftPart* and *RP: RightPart* which split a tuple into

two disjoint tuples. The binary function "$\asymp$" represents the concatenation of two tuples.

Base case 1 : We prove $\underset{Y}{\exists} O_F[\langle\rangle, Y]$. If the proof succeeds to find a ground term $R1$ such that $O_F[\langle\rangle, R1]$, then we know that $F[\langle\rangle] = R1$.

Base case 2: Similarly, $F[a \frown \langle\rangle] = R2$.

Induction step: For arbitrary but fixed A_0, B_0 (satisfying I_F), we assume $\underset{Y}{\exists} O_F[A_0, Y]$ and $\underset{Y}{\exists} O_F[B_0, Y]$ and we prove $\underset{Y}{\exists} O_F[A_0 \asymp B_0, Y]$. We Skolemize the assumptions by introducing two new constants Y_1 and Y_2 for the two existential Y. The existential variable Y in the goal becomes a metavariable Y^*. If the proof succeeds to find a witness $Y^* = T[A_0, B_0, Y_1, Y_2]$, then we know that $F[A \asymp B] = T[A, B, F[A], F[B]]$.

Algorithm extraction: Finally the algorithm is expressed as:

$$F[X] = \begin{cases} R1, \text{ if } X = \langle\rangle \\ R2, \text{ if } X = a \frown \langle\rangle \\ T[LP[X], RP[X], F[LP[X]], F[RP[X]]], \\ \qquad \text{if } X \neq \langle\rangle, X \neq (a \frown \langle\rangle) \end{cases}$$

Ind-2-Vars: The head–tail induction principle for two variables:

$$\Big(P[\langle\rangle, \langle\rangle] \bigwedge \underset{a,X}{\forall}(P[X, \langle\rangle] \Longrightarrow P[a \frown X, \langle\rangle]) \bigwedge \underset{b,Y}{\forall}(P[\langle\rangle, Y] \Longrightarrow P[\langle\rangle, b \frown Y]) \bigwedge \underset{a,b,X,Y}{\forall}(P[X, Y] \Longrightarrow P[a \frown X, b \frown Y])\Big) \Longrightarrow \underset{X,Y}{\forall} P[X, Y]$$

Here $P[X, \langle\rangle] : I_F[X] \Longrightarrow \underset{Z}{\exists} O[X, Z]$,
$P[\langle\rangle, Y] : I_F[Y] \Longrightarrow \underset{Z}{\exists} O[Y, Z]$,
$P[X, Y] : (I_F[X] \bigwedge I_F[Y]) \Longrightarrow \underset{Z}{\exists} O[X \asymp Y, Z]$

Base case 1: We prove $\underset{Z}{\exists} O_F[\langle\rangle, Z]$. If the proof succeeds to find a ground term $R1$ such that $O_F[\langle\rangle, R1]$, then we know that $F[\langle\rangle, \langle\rangle] = R1$.

Base case 2 and 3: Similarly $F[X, \langle\rangle] = R2$ and $F[\langle\rangle, Y] = R3$.

Induction step: For arbitrary but fixed X_0, Y_0 (satisfying I_F), we assume $\underset{Z}{\exists} O_F[X_0, Z]$ and $\underset{Z}{\exists} O_F[Y_0, Z]$ and we prove $\underset{Z}{\exists} O_F[(a \frown X_0) \asymp (b \frown Y_0), Z]$. We Skolemize the assumptions by introducing two new constants Z_1 and Z_2 for the two existential Z. The existential variable Z in the goal becomes a metavariable Z^*. If the proof succeeds to find a witness $Z^* = T[X_0, Y_0, Z_1, Z_2]$, then we know that $F[a \frown X, b \frown Y] = T[X, Y, F[X], F[Y]]$.

Algorithm extraction: Finally the algorithm is expressed as:

$$F[X, Y] = \begin{cases} R1, & \text{if } X = \langle\rangle, Y = \langle\rangle \\ R2, & \text{if } X \neq \langle\rangle, Y = \langle\rangle \\ R3, & \text{if } X = \langle\rangle, Y \neq \langle\rangle \\ T[X, Y, F[X], F[Y]], & \text{if } X \neq \langle\rangle, Y \neq \langle\rangle \end{cases}$$

Ind-Special: is a refinement of **Ind-2-Vars**.

Since $(a \frown X) \asymp Y$ and $X \asymp (b \frown Y)$ are smaller than $(a \frown X) \asymp (b \frown Y)$ we use the induction principle:

$$\Big(\underset{b,Y}{\forall}(P[\langle\rangle, Y] \Longrightarrow P[\langle\rangle, b \frown Y]) \bigwedge \underset{a,X}{\forall}(P[X, \langle\rangle] \Longrightarrow P[a \frown X, \langle\rangle]) \bigwedge \underset{a,b,X,Y}{\forall}((P[a \frown X, Y] \bigwedge P[X, b \frown Y]) \Longrightarrow P[a \frown X, b \frown Y])\Big) \Longrightarrow \underset{X,Y}{\forall} P[X, Y]$$

where
$P[a \frown X, Y] : (I_F[(a \frown X)] \bigwedge I_F[Y]) \Longrightarrow \underset{Z}{\exists} O[(a \frown X) \asymp Y, Z]$,
$P[X, b \frown Y] : (I_F[X] \bigwedge I_F[(b \frown Y)]) \Longrightarrow \underset{Z}{\exists} O[X \asymp (b \frown Y), Z]$.

Base case 1 and 2: Are the same with *Base case 3 and 2* from the previous induction **Ind-2-Vars**.

Induction Step: For arbitrary but fixed a, b, and X_0, Y_0 (satisfying I_F), we assume $\underset{Z}{\exists} O_F[(a \frown X_0) \asymp Y_0, Z]$ and $\underset{Z}{\exists} O_F[X_0 \asymp (b \frown Y_0), Z]$ and we prove $\underset{Z}{\exists} O_F[(a \frown X_0) \asymp (b \frown Y_0), Z]$. We Skolemize the assumptions by introducing two new constants Z_3 and Z_4 for the two existential Z. The existential variable Z in the goal becomes a metavariable Z^*. If the proof succeeds to find a witness $Z^* = T[a, b, X_0, Y_0, Z_3, Z_4]$, then we know that $F[a \frown X, b \frown Y] = T[a, b, X, Y, F[a \frown X, Y], F[X, b \frown Y]]$.

III. Reasoning

In this section we describe the design of our prover, namely the inference rules: first we use a back-chaining mechanism in *Prolog style* and then we add various specific prove-solve methods for the theory of tuples, functions and options for the user.

A. Prolog style (Back-chaining)

In this approach the prover will try to do matching on the first conjunct of the goal. For instance in order to prove the conjunction of two formulae containing the same metavariable $Goal1[Y^*] \wedge Goal2[Y^*]$, first it tries to find the witness term T for $Goal1[Y^*]$ and then tries to prove $Goal2[T]$ (a backtracking mechanism is implemented here).

Matching and unification:

If both assumptions contain variables and the goal contain variables or meta-variables, then unification takes place. (Otherwise it is matching.)

The most general case is unifying a conjunctive goal with an universal assumed implication. We illustrate this by examples. For instance if the assumption is $(FormLHS \Longrightarrow FormRHS)$ and the goal is $Goal1 \wedge Goal2$ (the assumption can be quantified and the goal can contain metavariables) it first tries to do matching on $Goal1$ with $FormRHS$ of the assumed implication and if the matching is done we obtain a substitution, then it is sufficient to prove $FormLHS \wedge Goal2$(under the substitution). $FormLHS$ can be a conjunction of formulae. Or if the assumption is $(FormLHS \Longrightarrow FormRHS)$ and the goal is $Goal$ (the assumption can be quantified and the goal can contain metavariables) it first tries to do matching of $Goal$ with $FormRHS$ of the assumed implication and if the matching is done, then it is sufficient to prove $FormLHS$. If after matching we obtain a substitution, then the new goal is to prove $FormLHS$(under the substitution).

Particular cases are: the goal can be unique or a conjunction of terms, can be ground (does not contain variables or metavariables) or can contain variables or metavariables; the assumption can be unique or an implication of terms, can be ground (does not contain variables or metavariables) or can contain variables.

For the cases when the goal is not ground and/or the assumption is not ground, after unification/matching we obtain substitutions for variables and for metavariables. Variables are substituted with priority, for e.g. if the goal (which contains metavariables) is $S[a^* \smile (b_0 \smile C_0)]$, where a^* is a metavariable, b_0, C_0 are constants and $\underset{a,b,X}{\forall}((S[b \smile X] \wedge a \leq b) \Longrightarrow S[a \smile (b \smile X)])$ is the assumption, by matching we obtain the substitutions $\{a \rightarrow a^*, b \rightarrow b_0, X \rightarrow C_0\}$. The next step is to find a correct substitution for a^*.

In some cases we need to transform variables into metavariables. For e.g. if the goal is $(a_0 \lhd Y^*) \wedge Goal2[Y^*]$ and the assumption is $\underset{a,A}{\forall}(a \lhd F[a, A])$ then by unification of the first conjunct of the goal with universal assumption we find substitutions $\{a \rightarrow a_0, Y^* \rightarrow F[a_0, A^*]\}$. The variable A also needs to be instantiated and by unification we do not find any value for it, therefore it is transformed into metavariable, it becomes our witness and we need to find a substitution for it.

B. Special inference rules

In some cases the inference rules presented above are not sufficient in order to obtain a correct term for our witness. Therefore and also in order to avoid searching and a proof tree with many branches we add specific prove-solve methods:

SR1: *Rewrite goal (or a part of the goal) using assumption* $LHS \approx RHS$ ("$\approx$" stands for having the same elements.)

We explain this rule using an example. The assumption is: $a \smile A \approx B$ and the goal is: $a \smile (b \smile (A \asymp C)) \approx Z$. For the left hand side of the assumption we have the set of simple symbols: $\{a\}$, the set of tuple symbols: $\{A\}$; for the right hand side of the assumption the set of simple symbols is: $\{\}$, the set of tuple symbols is: $\{B\}$. For the goal the set of simple symbols is: $\{a, b\}$, the set of tuple symbols is: $\{A, C\}$. We check if the sets of simple and tuple symbols of the left hand side of the assumption are included in the sets of simple and tuple symbols of the goal. In our case $\{a\}$ is included in $\{a, b\}$ and $\{A\}$ is included in $\{A, C\}$. After this, by subtraction we obtain the new goal $b \smile (B \asymp C) \approx Z$.

SR2: In the case of a conjunctive metavariable goal, for e.g. the goal is $b \smile (a \smile A) \approx Z^* \wedge$ *IsSorted*$[Z^*]$ and we rewrite the first conjunct of the goal using an assumption $a \smile A \approx B$ then the new goal will be *IsSorted*$[b \smile B]$. Here it makes two steps, first it rewrites the first conjunct of the goal and finds a witness and second it replaces the new term into the second conjunct of the goal and tries to prove the new goal. This rule eliminates the need of using the proposition *"reflexivity"*.

SR3: *Rewrite special ground goal (or a part of the goal) of the form* $LHS \precsim RHS$

We use the notation "$\leq$" for the ordering between elements. Likewise $a \precsim A$ means that the element a is smaller than all the elements from the tuple A.

For a goal of the form $LHS \precsim RHS$, RHS can be replaced by an equivalent (by $\approx$) expression. For instance if the goal is $a \precsim Y_1$ and the assumption is $b \smile Y_0 \approx Y_1$, then the new goal is $a \precsim b \smile Y_0$.

When RHS is an expression we split the goal into simpler inequalities. For instance, the goal $b \precsim a \smile (A \asymp B)$ becomes $b \precsim B \wedge b \precsim A \wedge b \leq a$;

SR4: *Reduce the goal using an assumption*

If a part G_1 of a conjunctive goal is a consequence of another part G_2 of the goal and the assumptions, then the part G_1 is deleted from the goal. For instance, when the goal is $a \precsim Y_4 \wedge a \leq b$ and among the assumptions we have $b \precsim Y_4$, then we can reduce the goal to $a \leq b$, because $a \precsim Y_4$ is a consequence of $a \leq b$ and $b \precsim Y_4$ (by the reverse transitivity axiom $\underset{a,b,X}{\forall}(b \precsim X \wedge a \leq b \Longrightarrow a \precsim X$.)

SR5: *Transform ground assumptions – Forward Reasoning*

When in the assumption we have an expression of the form *IsSorted*$[expr]$, if $expr$ is a tuple expression and not a symbol, then we transform the formula into atomic formulae (e.g. if the assumption is *IsSorted*$[a \smile A_0]$, then the formula is transformed into $a \precsim A_0$ and *IsSorted*$[A_0]$).

SR6: *Transform ground goal – Backward Reasoning*

When the goal is a ground expression of the form *IsSorted*$[expr]$, if $expr$ is a tuple expression and not a symbol, then we transform the formula into atomic formulae (e.g. if the goal is *IsSorted*$[a \smile A_0]$, then it is transformed into *IsSorted*$[A_0]$ and $a \precsim A_0$).

The purpose of transformation and rewriting is multifold: to overcome the search space explosion, to simplify

the formulae that occur into the proof until we obtain only atomic formulae without loosing information, therefore we need to take care about the saturation of the formulae obtained as assumptions and as goals. And after we obtain the saturated sets of formulae it is easier to see what formulae occur in the assumption and eliminate them from the goals.

IV. Case studies

In this section we describe the automatically generated proofs and algorithms by using the prover described in section III (full details of the proofs are presented in [12]). The formulae (definitions, propositions, etc.) are written like in the *Theorema* system. The typing is not explicit: some symbols and expressions are tuples, but this is determined unambiguously by the context in which they appear. The monotone and repetitive style of the proof text should be excused on grounds of being computer produced.

The Problem of Sorting

The *Problem Specification* is

$$I_{\mathrm{F}}[X] : \text{True}$$
$$O_{\mathrm{F}}[X,Y] : \left\{ \begin{array}{c} X \approx Y \\ \mathit{IsSorted}[Y] \end{array} \right.$$

and we have to prove:

$$\mathbf{Proposition}\Big[\text{“Problem of Sorting”}, \underset{X}{\forall}\underset{Y}{\exists}(X \approx Y \wedge IsSorted[Y])\Big]$$

Common Knowledge Base

$$\mathbf{Definition}\Big[\text{“}\approx\text{”}, \text{any}[a,X,Y], \langle\rangle \approx \langle\rangle, ((a \frown X) \approx Y) \Longrightarrow ((a \triangleleft Y) \wedge X \approx \mathit{dfo}[a,Y])\Big]$$

We do not use explicitly the definitions of “$\triangleleft$” (occurs in) and “*dfo*” (delete first occurence) (see [12]), only the properties of it.

$$\mathbf{Definition}\Big[\text{“}\mathit{IsSorted}\text{”}, \text{any}[a,b,X], \mathit{IsSorted}[\langle\rangle], \mathit{IsSorted}[a \frown \langle\rangle], (\mathit{IsSorted}[b \frown X] \wedge (a \leq b)) \Longrightarrow \mathit{IsSorted}[a \frown (b \frown X)]\Big]$$

$$\mathbf{Definition}\Big[\text{“}\mathit{concatenation}\text{”}, \text{any}[a,X,Y], \langle\rangle \asymp X = X, X \asymp \langle\rangle = X, (a \frown X) \asymp Y = (a \frown (X \asymp Y))\Big]$$

$$\mathbf{Proposition}[\text{“}\mathit{reflexivity}\text{”}, \text{any}[X], X \approx X]$$

The problem specification and the properties above are essentially the same as in [3], but without sequence variables.

A. Synthesis of Insertion-Sort Algorithm

We use the *Head-Tail-Induction1* principle.

For proving the proposition *“Problem of Sorting ”* we use Definition “$\approx$”, Definition *“IsSorted”* and also the assumptions:

$$\mathbf{Proposition}\Big[\text{“}\mathit{dfo1}\text{”}, \text{any}[a,S,M], ((a \triangleleft M) \wedge S \approx \mathit{dfo}[a,M]) \Longrightarrow ((a \frown S) \approx M)\Big]$$

$$\mathbf{Proposition}\Big[\text{“}\mathit{IsElem\ in\ Insertion}\text{”}, \text{any}[a,A], (a \triangleleft \mathit{Insertion}[a,A])\Big]$$

$$\mathbf{Proposition}\Big[\text{“}\mathit{The\ same\ elements\ in\ Ins}\text{”}, \text{any}[a,V,T], ((V \approx T) \Longrightarrow (V \approx \mathit{dfo}[a, \mathit{Insertion}[a,T]]))\Big]$$

$$\mathbf{Proposition}\Big[\text{“}\mathit{Sorting\ using\ Insertion}\text{”}, \text{any}[a,B], \mathit{IsSorted}[B] \Longrightarrow \mathit{IsSorted}[\mathit{Insertion}[a,B]]\Big]$$

We prove Proposition *“Problem of Sorting”* by Induction on *X*.

1. Induction Base: We have to prove

$$(1)\quad \langle\rangle \approx \mathrm{Y3}_0^* \wedge \mathit{IsSorted}[\mathrm{Y3}_0^*]$$

In order to prove (1), by Definition “$\approx$”: 1 using substitution $\{\mathrm{Y3}_0^* \rightarrow \langle\rangle\}$, it is sufficient to prove:

$$(2)\quad \mathit{IsSorted}[\langle\rangle]$$

Formula (2) is true because it is identical to Definition *“IsSorted”*: 1 .

2. Induction Step: We assume

$$(3)\quad \mathit{X1}_0 \approx \mathit{Y4}_0,$$
$$(4)\quad \mathit{IsSorted}[\mathit{Y4}_0],$$

and find witness such that:

$$(5)\quad \mathit{a2}_0 \frown \mathit{X1}_0 \approx \mathrm{Y3}_0^* \wedge \mathit{IsSorted}[\mathrm{Y3}_0^*]$$

In order to prove (5), by Proposition *“dfo1”* using substitution $\{a \rightarrow \mathit{a2}_0,\ S \rightarrow \mathit{X1}_0,\ M \rightarrow \mathrm{Y3}_0^*\}$, it is sufficient to prove:

$$(6)\quad \mathit{a2}_0 \triangleleft \mathrm{Y3}_0^* \wedge\ \mathit{X1}_0 \approx \mathit{dfo}[\mathit{a2}_0, \mathrm{Y3}_0^*] \wedge \mathit{IsSorted}[\mathrm{Y3}_0^*]$$

In order to prove (6), by Proposition “*IsElem in Insertion*” using substitution $\{a \rightarrow \mathit{a2}_0,\ \mathrm{Y3}_0^* \rightarrow \mathit{Insertion}[\mathit{a2}_0,\ A_0^*]\}$, it is sufficient to prove:

$$(7)\quad \mathit{X1}_0 \approx \mathit{dfo}[\mathit{a2}_0,\ \mathit{Insertion}[\mathit{a2}_0,\ A_0^*]] \wedge \mathit{IsSorted}[\mathit{Insertion}[\mathit{a2}_0,\ A_0^*]]$$

In order to prove (7), by Proposition “*The same elements in Ins*” using substitution $\{V \rightarrow \mathit{X1}_0,\ a \rightarrow \mathit{a2}_0,\ T \rightarrow A_0^*\}$, it is sufficient to prove:

$$(8)\quad \mathit{X1}_0 \approx A_0^* \wedge \mathit{IsSorted}[\mathit{Insertion}[\mathit{a2}_0,\ A_0^*]]$$

In order to prove (8), by (3) using substitution $\{A_0^* \rightarrow Y4_0\}$, it is sufficient to prove:

$$(9)\ \ IsSorted[Insertion[a2_0,\ Y4_0]]$$

In order to prove (9), by Proposition "Sorting using Insertion" using substitution $\{a \rightarrow a2_0,\ B \rightarrow Y4_0\}$, it is sufficient to prove:

$$(10)\ \ IsSorted[Y4_0]$$

Formula (10) is true because it is identical to (4).

The extracted algorithm from the proof is:

$$\underset{a,X}{\forall} \left(\begin{array}{c} F[\langle\rangle] = \langle\rangle \\ F[a \smile X] = Insertion[a,\ F[X]] \end{array} \right)$$

For this case study it is sufficient to apply the *Prolog style*. Another version of this proof is presented in [12].

B. Synthesis of Insertion Algorithm

$$\textbf{Proposition}\Big[\text{"reduced Problem-Insertion "},\ \underset{a,T,IsSorted[T]}{\forall}\ \underset{Y}{\exists}(T \approx Y \wedge IsSorted[Y])\Big]$$

Note that our previous problem reduces into a simpler problem because the input condition is extended by adding the fact that T *IsSorted*. Our problem now is to find an algorithm given an element and a sorted tuple it will return a sorted tuple. For details about reducing the problem please see the description of the "cascade" method in [10] and also the case study made by hand in [13].

In order to prove the Proposition "*reduced Problem-Insertion*" we use the *Head-Tail-Induction2* induction principle applied on the tail of the tuple.

We prove Proposition *"reduced Problem-Insertion"* by Induction on *T* under the assumptions: Definition *"IsSorted"* and Proposition *"reflexivity"*.

1. (The Induction Base: is straight forward and we omit the presentation.)

2. Induction Step: We assume

$$(3)\ \ a2_0 \smile T1_0 \approx Y4_0,\ (4)\ \ IsSorted[Y4_0],$$
$$(5)\ \ IsSorted[b2_0 \smile T1_0],$$

and find witness such that:

$$(6)\ \ a2_0 \smile (b2_0 \smile T1_0) \approx \mathrm{Y3}_0^* \wedge \mathrm{IsSorted}[\mathrm{Y3}_0^*]$$

Proof 1: *in Prolog style*

1) Case $a \leq b$: In order to prove (6), by Proposition *"reflexivity"* and using substitution $\{X \rightarrow a2_0 \smile (b2_0 \smile T1_0),\ \mathrm{Y3}_0^* \rightarrow a2_0 \smile (b2_0 \smile T1_0)\}$, it is sufficient to prove:

$$(7)\ \ IsSorted[a2_0 \smile (b2_0 \smile\ T1_0)]$$

In order to prove (7), by Definition *"IsSorted": 3* using substitution $\{a \rightarrow a2_0,\ b \rightarrow b2_0,\ X \rightarrow T1_0\}$, it is sufficient to prove:

$$(8)\ \ IsSorted[b2_0 \smile T1_0] \wedge a2_0 \leq b2_0$$

In order to prove (8), by (5) it is sufficient to prove:

$$(9)\ \ a2_0 \leq b2_0$$

When we reach a goal like (9) which is very simple, but cannot be proved or disproved, we accept it as the conditional assumption on the corresponding branch.

We also need the proof for the other case when $b \leq a$.

2) Case $b \leq a$: We use in the knowledge base: Proposition *"reflexivity"*, Definition *"IsSorted"* and also the propositions:

$$\textbf{Proposition}\Big[\text{"}\precsim \textit{sorting"}, \mathrm{any}[a, Z],\ (IsSorted[Z] \wedge (a \precsim Z)) \Longrightarrow IsSorted[a \smile Z]\Big]$$

$$\textbf{Proposition}\Big[\textit{"sorting} \smile \textit{"}, \mathrm{any}[a, b, X],\ (IsSorted[a \smile X] \wedge b \leq a) \Longrightarrow (b \precsim a \smile X)\Big]$$

In order to prove (6) by (3) using the witness $\{\mathrm{Y3}_0^* \rightarrow b2_0 \smile Y4_0\}$, it is sufficient to prove:

$$(7)\ \ IsSorted[b2_0 \smile\ Y4_0]$$

Note at this step the difference between the previous case, here instead of using the proposition *"reflexivity"* it rewrites the goal using equivalence assumption, obtain a term for the witness and continue with the rest of the goal under the substitution.

We proceed in a similar way and we reach the goal:

$$(12)\ \ IsSorted[T1_0] \wedge a2_0 \precsim T1_0 \wedge b2_0 \leq a2_0$$

It is possible in principle to perform the proof by backchaining (we are able to build the proof by hand), however the search space is too big and the automatic proof does not succeed in a reasonable time. Therefore we apply some special inference rules.

Proof 2: *Apply special inference rules*

We use in the knowledge base Proposition *"reflexivity"* and Definition *"IsSorted"*.

We transform the assumption (5) into (apply **SR5**):

$$(9)\ \ b2_0 \precsim T1_0$$
$$(10)\ \ IsSorted[T1_0],$$

1) Case $a \leq b$:

In order to prove (6), by Proposition *"reflexivity"* and using substitution $\{X \rightarrow a2_0 \smile (b2_0 \smile T1_0),\ \mathrm{Y3}_0^* \rightarrow a2_0 \smile (b2_0 \smile T1_0)\}$, it is sufficient to prove:

$$(11)\ \ IsSorted[a2_0 \smile (b2_0 \smile\ T1_0)]$$

We transform our goal into proving (apply **SR6**):

$$(12)\ \ IsSorted[T1_0],\ (13)\ \ b2_0 \precsim T1_0,\ (14)\ \ a2_0 \leq b2_0$$

Formulae (12) and (13) are identical with our assumptions (10), respectively (9) so our new goal is:

$$(14)\ \ a2_0 \leq b2_0$$

Similarly to the previous case study the goal (14) becomes the conditional assumption on this branch.

2) Case $b \leq a$

In order to prove (6) by (3) and using the witness $\{\mathrm{Y3}_0^* \rightarrow b2_0 \smile Y4_0\}$ it suffices to prove (apply **SR1** and **SR2**):

$$(11)\ \ IsSorted[b2_0 \smile Y4_0]$$

We transform our goal into proving (apply **SR6**):

$$(12)\ \ IsSorted[Y4_0],\ (13)\ \ b2_0 \precsim Y4_0$$

Formula (12) is identical with our assumption (4) so our new goal is:

$$(13)\ \ b2_0 \precsim Y4_0$$

In order to prove (13) using (3) it suffices to prove (apply **SR3**):

$$(14)\ \ b2_0 \precsim a2_0 \smile T1_0$$

We split our goal (14) into (apply **SR3**):

$$(15)\ \ b2_0 \precsim T1_0 \wedge b2_0 \leq a2_0$$

The first conjunct of (15) is identical with our assumption (9) so our new goal is:

$$(16)\ \ b2_0 \leq a2_0$$

which becomes the conditional assumption on this branch.

The extracted algorithm from the proof is:

$$\underset{a,b,X}{\forall} \left(\begin{array}{c} F[a,\ \langle\rangle] = a \smile \langle\rangle \\ F[a,\ b \smile Tail[X]] = \begin{cases} (a \smile (b \smile Tail[X])), \text{if } a \leq b \\ b \smile F[a, Tail[X]],\ \text{if } b \leq a \end{cases} \end{array} \right)$$

C. Synthesis of Merge-Sort Algorithm

We start to prove the Proposition *"Problem of Sorting"* using the induction principle *Divide-And-Conquer-Induction* by using the assumptions: Definition *"IsSorted"*, Proposition *"reflexivity"* and also the propositions:

$$\textbf{Proposition}\Big[\text{"Concat equiv Merge"}, \text{any}[A,B,V,T], (A \approx V \wedge B \approx T) \Longrightarrow ((A \asymp B) \approx Merge[V,T])\Big]$$

$$\textbf{Proposition}\Big[\text{"Merge 1"}, \text{any}[V,T], (IsSorted[V] \wedge IsSorted[T]) \Longrightarrow IsSorted[Merge[V,T]]\Big]$$

We prove Proposition *"Problem of Sorting"* by Induction on X.

1. Induction Base1: Similar with induction base from Insertion-Sort, we find witness $\mathrm{Y3}_0^* \rightarrow \langle\rangle$

2. Induction Base2: Similar with induction base from Insertion, we find witness $\mathrm{Y3}_0^* \rightarrow a2_0 \smile \langle\rangle$

3. Induction Step: We assume

$$(5)\ \ A1_0 \approx Y4_0,\ (6)\ \ IsSorted[Y4_0],\ (7)\ \ B1_0 \approx Y5_0,\ (8)\ \ IsSorted[Y5_0],$$

and find witness such that:

$$(9)\ \ A1_0 \asymp B1_0 \approx \mathrm{Y3}_0^* \wedge IsSorted[\mathrm{Y3}_0^*]$$

In order to prove (9), by Proposition *"Concat equiv Merge"* using substitution $\{A \rightarrow A1_0,\ B \rightarrow B1_0,\ \mathrm{Y3}_0^* \rightarrow Merge[V_0^*, T_0^*]\}$, it is sufficient to prove:

$$(10)\ \ A1_0 \approx V_0^* \wedge B1_0 \approx T_0^* \wedge IsSorted[Merge[V_0^*, T_0^*]]$$

In order to prove (10), by (5) using substitution $\{V_0^* \rightarrow Y4_0\}$, it is sufficient to prove:

$$(11)\ \ B1_0 \approx T_0^* \wedge IsSorted[Merge[Y4_0, T_0^*]]$$

In order to prove (11), by (7) using substitution $\{T_0^* \rightarrow Y5_0\}$, it is sufficient to prove:

$$(12)\ \ IsSorted[Merge[Y4_0, Y5_0]]$$

In order to prove (12), by Proposition *"Merge 1"* using substitution $\{V \rightarrow Y4_0,\ T \rightarrow Y5_0\}$, it is sufficient to prove:

$$(13)\ \ IsSorted[Y4_0] \wedge IsSorted[Y5_0]$$

In order to prove (13), by (6) it is sufficient to prove:

$$(14)\ \ IsSorted[Y5_0]$$

Formula (14) is true because it is identical to (8).

The extracted algorithm from the proof is:

$$\underset{a,A,B}{\forall} \left(\begin{array}{c} F[\langle\rangle] = \langle\rangle \\ F[a \smile \langle\rangle] = a \smile \langle\rangle \\ F[A \asymp B] = Merge[F[A], F[B]] \end{array} \right)$$

Therefore also for this case study, it is sufficient to apply the inference rules in *Prolog style* in order for the proof to succeed.

D. Synthesis of Merge Algorithm

1) Synthesis of Merge Algorithm-reduced problem 1: This is the case when we do not have the definition and the properties for the function *"Merge"* into the knowledge base and we synthesize it. The problem reduces into proving:

$$\textbf{Proposition}\Big[\ \text{"reduced Problem1"}, \underset{Y1,\,Y2,\,IsSorted[Y1],\,IsSorted[Y2]}{\forall} \underset{Z}{\exists} \Big((Y1 \asymp Y2) \approx Z \wedge \text{IsSorted}[Z]\Big)\Big]$$

We use the induction principle *Ind-2-Vars*.

The three base cases of the induction principle are very simple and we obtain them in similar way as we presented. We omit the presentation here, for details please see [12].

2) Synthesis of Merge Algorithm-reduced problem 2: Because the proof of the induction step does not succeed we reduce our problem into proving the following:

$$\textbf{Proposition}\Big[\ \text{"reduced Problem2"}, \underset{a,\,b,\,Y1,\,Y2}{\forall} \underset{Z}{\exists} \Big((a \smile Y1) \asymp (b \smile Y2) \approx Z \wedge \text{IsSorted}[Z]\Big)\Big]$$

In order to prove the Proposition *"reduced Problem2"* we use *Ind-Special*. We assume: *IsSorted*$[a \frown Y_1]$ and *IsSorted*$[b \frown Y_2]$.

Prove Proposition *"reduced Problem2"* under the assumptions: Definition *"IsSorted"*, Definition *"concatenation"*, Proposition *"$\precsim$ sorting"* and also:

$$\textbf{Proposition}\Big[\textit{"the smallest element2"}, \text{any}[a, X, Y], (a \precsim X \wedge a \precsim Y) \Longrightarrow (a \precsim (X \asymp Y))\Big]$$

$$\textbf{Proposition}\Big[\text{"}\leq\text{"}, \text{any}[a, b, X], (\text{IsSorted}[b \frown X] \wedge (a \leq b)) \Longrightarrow (a \precsim (b \frown X))\Big]$$

$$\textbf{Proposition}\Big[\textit{"ineq of sort"}, \text{any}[a, A], \text{IsSorted}[a \frown A] \Longrightarrow (a \precsim A)\Big]$$

We rewrite the formula Proposition *"reduced Problem2"* using Definition *"concatenation: 3"* and we obtain a new goal:

$$(1) \quad \underset{a,\,b,\,Y1,\,Y2}{\forall} \; \underset{Z}{\exists} \Big(a \frown (Y1 \asymp b \frown Y2) \approx Z \wedge \mathit{IsSorted}[\mathrm{Z}]\Big)$$

We prove (1) by Induction on $\{Y1, Y2\}$.

1. Induction Step: We assume

$$(2)\ \mathit{IsSorted}[a2_0 \frown Y3_0],\ (3)\ \ Y3_0 \asymp b2_0 \frown Y4_0 \approx Z2_0,$$
$$(4)\ \mathit{IsSorted}[Z2_0],\ (5)\ \mathit{IsSorted}[b2_0 \frown Y4_0],$$
$$(6)\ \ a2_0 \frown (Y3_0 \asymp Y4_0) \approx Z3_0,\ (7)\ \ \mathit{IsSorted}[Z3_0],$$

and find witness such that:

$$(8)\ \ a2_0 \frown (Y3_0 \asymp b2_0 \frown Y4_0) \approx \mathrm{Z1}_0^* \wedge \mathit{IsSorted}[\mathrm{Z1}_0^*]$$

Proof 1: *in Prolog style*

1) Case $a \leq b$

In order to prove (8) by (3) using the witness $\{\mathrm{Z1}_0^* \rightarrow a2_0 \frown Z2_0\}$ it suffices to prove:

$$(9)\ \ \mathit{IsSorted}[a2_0 \frown Z2_0]$$

The proof proceeds in *Prolog style* using back-chaining and we arrive at the goal:

$$(17)\ \ a2_0 \leq b2_0$$

which becomes the conditional assumption on this branch.

2) Case $b \leq a$

We prove Proposition *"reduced Problem2"* under the assumptions: Definition *"IsSorted"*, Definition *"concatenation"*, Proposition *"$\precsim$ sorting"*, Proposition *"ineq of sort"* and also:

$$\textbf{Proposition}\Big[\textit{"the smallest element4"}, \text{any}[a, X, Y], (a \precsim Y \wedge a \precsim X) \Longrightarrow (a \precsim (X \asymp Y))\Big]$$

$$\textbf{Proposition}\Big[\textit{"}\leq\textit{2"}, \text{any}[a, b, X], (b \precsim X \wedge b \leq a \Rightarrow b \precsim a \frown X)\Big]$$

$$\textbf{Proposition}[\text{" } \precsim \wedge \leq \text{ "}, \text{any}[a, b, X], (\mathit{IsSorted}[b \frown (a \frown X)] \wedge (b \leq a)) \Longrightarrow (b \precsim X)]$$

In order to prove (8) by (6) using the witness $\{\mathrm{Z1}_0^* \rightarrow b2_0 \frown Z3_0\}$ it suffices to prove:

$$(9)\ \ \mathit{IsSorted}[b2_0 \frown Z3_0]$$

The proof proceeds in a similar way and we arrive at the goal:

$$(22)\ \ b2_0 \leq a2_0$$

which becomes the conditional assumption on this branch.

Proof 2: *Apply special inference rules*

In this case we use only the Definition *"concatenation"* and no other propositions at all.

We transform the assumption (2) into (apply **SR5**):

$$(9)\ \ a2_0 \precsim Y3_0$$
$$(10)\ \ \mathit{IsSorted}[Y3_0],$$

We transform the assumption (5) into (apply **SR5**):

$$(11)\ \ b2_0 \precsim Y4_0$$
$$(12)\ \ \mathit{IsSorted}[Y4_0],$$

1) Case $a \leq b$

In order to prove (8) by (3) and using the witness $\mathrm{Z1}_0^* \rightarrow a2_0 \frown Z2_0$ it suffices to prove:

$$(13)\ \ \mathit{IsSorted}[a2_0 \frown Z2_0]$$

Similar with *Proof 2, Case 2* from Insertion we reach the new goal:

$$(18)\ \ a2_0 \precsim Y4_0 \wedge a2_0 \leq b2_0$$

In order to prove our special goal (18) by using the assumption (11) and by applying the reversed transitivity it is sufficient to prove (apply **SR4**):

$$(19)\ \ a2_0 \leq b2_0$$

which becomes the conditional assumption on this branch.

2) Case $b \leq a$

In a similar way in order to prove (8) by (6) we obtain the witness $\mathrm{Z1}_0^* \rightarrow b2_0 \frown Z3_0$ and the conditional assumption for this branch is $b2_0 \leq a2_0$.

The extracted algorithm from the proof is:

$$\underset{a,b,Y1,Y2}{\forall} \left(\begin{array}{c} F[\langle\rangle, \langle\rangle] = \langle\rangle \\ F[Y1,\ \langle\rangle] = Y1 \\ F[\langle\rangle,\ Y2] = Y2 \\ F[Y1,\ Y2] = \begin{cases} a \frown F[Y1, b \frown Y2], & \text{if } a \leq b \\ b \frown F[a \frown Y1, Y2], & \text{if } b \leq a \end{cases} \end{array} \right)$$

V. Conclusion and Further Work

In this paper we present some interesting proof techniques for automated reasoning on tuples in proof based synthesis. We describe the method that we use in order to construct an

inductive proof of the fact that for each input there exists a sorted tuple. The existential variable from the goal becomes a metavariable (witness) which we solve during the proof by applying some inference rules.

In the first type of proofs in *Prolog style* (we apply a backchaining mechanism) in order to prove the conjecture we use some definitions and some propositions from the knowledge base, but in some cases this approach of constructing proofs it is not enough in order for the proof to succeed and in order to find a correct witness term during the proof. Therefore, we also add some special inference rules (various specific prove-solve methods (equivalence rules, inequality rules) introduced in order to overcome the search explosion by eliminating the rules with backtracking) based on the properties of tuples. By using this kind of special inference rules there is no need to use propositions and definitions and we develop a solving mechanism for algorithm synthesis based on a combination of backward and forward inferences which complements the *Prolog style* reasoning.

We show on some concrete examples the automated process of constructing proofs, synthesis and extraction of the algorithms from efficient proofs implemented in the *Theorema* system (like e.g. Insertion-Sort, Insertion, Merge-Sort, Merge) obtained by using different knowledge bases and by applying a different induction principle. Of course we cannot speculate on the generality of these techniques, based on the few case studies for known algorithms which are presented here. However, we achieve a high degree of automation on such proofs by using novel proof techniques, and this paves the way for further experiments and exploration of more sophisticated problems and algorithms.

The main goal of our work is to improve the methods for program synthesis. Besides the special proof techniques, this work also gives useful hints about finding appropriate induction principles for tuples, as well as for building the tuple theory (by constructing appropriate collections of properties which are necessary for reasoning about sorting).

We plan to extend this work by investigating further algorithms in a similar way (like e. g. selection sort, quick sort). By investigation of more and more cases we can improve our prover to be more general and to be able to synthesize and to extract a huge number of algorithms. The work can then proceed in the direction of proving the correctness and the consistence of the prover.

REFERENCES

[1] P. Audebaud and L. Chiarabini. New Development in Extracting Tail Recursive Programs from Proofs. In *LOPSTR 2009*.

[2] D. Basin, Y. Deville, P. Flener, A. Hamfelt, and Jorgen Fischer Nilsson. Synthesis of programs in computational logic. In *PROGRAM DEVELOPMENT IN COMPUTATIONAL LOGIC*, pages 30–65. Springer, 2004.

[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 Invention and Verification by Lazy Thinking. In *Analele Universitatii de Vest, Timisoara, Ser. Matematica - Informatica*, volume XLI, pages 41–70, 2003.

[5] B. Buchberger and A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. *Electr. Notes Theor. Comput. Sci.*, 93:24–59, 2004.

[6] 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.

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

[8] 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.

[9] J. Darlington. A Synthesis of Several Sorting Algorithms. *Acta Inf.*, 11:1–30, 1978.

[10] I. Dramnesc and T. Jebelean. A Case Study in Proof Based Synthesis of Sorting Algorithms. In *Analele Universitatii de Vest, Timisoara, Ser. Matematica - Informatica*, volume XLVIII, pages 47–58, 2010.

[11] I. Dramnesc and T. Jebelean. Proof Based Synthesis of Sorting Algorithms. Technical Report 10-17, RISC Report Series, University of Linz, Austria, 2010.

[12] I. Dramnesc and T. Jebelean. Automated Reasoning on Tuples - Case Studies in Proof Based Synthesis. Technical Report 11-08, RISC Report Series, University of Linz, Austria, 2011.

[13] I. Dramnesc, T. Jebelean, and A. Craciun. A Case Study in Systematic Exploration of Tuple Theory. In *SCSS 2010*, volume 1, pages 82–95. RISC-Linz Report Series No. 10-10.

[14] I. Kraan, D. Basin, and A. Bundy. Middle-Out Reasoning for Logic Program Synthesis. In *ICLP 1993*, pages 441–455. MIT Press.

[15] K. K. Lau and G. Wiggins. A Tutorial on Synthesis of Logic Programs from Specifications. In *ICLP 1994*, pages 11–14. MIT Press.

[16] D. R. Smith. A problem reduction approach to program synthesis. In *IJCAI 1983*, pages 32–36. Morgan Kaufmann Publishers Inc.

[17] D. R. Smith. *Top-down synthesis of divide-and-conquer algorithms*, pages 35–61. Morgan Kaufmann Publishers Inc., 1986.

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