Functional Program Verification with *Theorema*

Adrian Crăciun

Research Institute for Symbolic Computation, Hagenberg, Austria
acraciun@risc.uni-linz.ac.at

Abstract

Functional programs and their correctness statement can be expressed in the language of higher order predicate logic. We take as an example the algorithm of sorting by merging of tuples, show how this is expressed in the *Theorema* language, a version of higher order predicate logic. Then, by the "lazy thinking" top-down exploration paradigm, we prove the correctness of the algorithm, using the appropriate provers provided by the system.

Theorema is a system for the working mathematician. It provides support for the basic mathematical activities: proving, computing, solving and exploration of mathematical theories, everything in the same language, the *Theorema* language, which is a version of (higher order) predicate logic. The proving part of the system consists of a general higher order predicate prover and a collection of special provers, that implement proof techniques for various mathematical domains (set theory, sequences, etc.). The special provers of *Theorema* are combined in the frame of the user provers, that implement proof strategies. The user provers are called to prove theorems. For a description of the system, see [4], [6].

In *Theorema* the emphasis is on theory exploration, rather than on proving isolated theorems. For the chosen example, we will specify the theory, define the algorithm and prove it correct using the provers provided by the system.

The theory of tuples is expressed in terms of sequences. Sequence variables(see [2], [3]) are variables that can be instantiated by an arbitrary finite number of terms. In the *Theorema*, sequence identifiers are denoted by overbarred identifiers: $\overline{\mathtt{x}}$, $\overline{\mathtt{y}}$ ...

In terms of sequences, tuples are represented in the following way in *Theorema*:

$$\begin{array}{l}\texttt{Definition["is tuple", any[X],}\\ \quad\texttt{is-tuple[X]} \Leftrightarrow \underset{\overline{x}}{\exists}(\texttt{X}= \langle\overline{x}\rangle)]\end{array}$$

Tuples are objects that satisfy the is-tuple predicate, and in terms of sequences are represented with the help of the angle brackets $(\langle, \rangle)$.

Define in a functional manner the merge-sort algorithm. The argument of the function is a tuple and its return value is the sorted version of the argument:

$$\begin{array}{l}\texttt{Algorithm["merge-sort", any[is-tuple[A]],}\\ \quad\texttt{stmg[A]:=}\left\{\begin{array}{lll}\texttt{A} & \Leftarrow & |\texttt{A}| \leq 1\\ \texttt{mg[stmg[lsp[A]], stmg[rsp[A]]]} & \Leftarrow & \texttt{otherwise}\end{array}\right. \texttt{]}\end{array}$$

In the above, **stmg** is the merge sort function, **mg** is the function definition for the merging of tuples, **lsp**, **rsp** are the left split and the right split of a tuple,$|\ldots|$ represents the length of a tuple.

The correctness statement says that the result of the application of the merge-sort function is a tuple, which is sorted and is a permuted version of the argument:

$$\begin{array}{l}\texttt{Proposition["correctness of merge-sort", any[is-tuple[A]],}\\ \quad\texttt{is-tuple[stmg[A]]}\\ \quad\texttt{stmg[A]}\approx\texttt{ A}\\ \quad\texttt{is-sorted[stmg[A]]] ,}\end{array}$$

where $\approx$ stands for the predicate is-permuted-version-of.

Now for the proof of the correctness. We will use the sequence user provers, which are a combinations of the sequence basic provers (arbitrary but fixed, case distinction, structural and well-founded induction for sequences), predicate logic in natural style, case distinction for case definitions (like in the above definition of merge−sort).

Proceed by lazy top-down exploration ("lazy thinking paradigm", introduced by B. Buchberger in [1], also described in [5]) in the theory of tuples. Try to prove the result by deliberately not providing any knowledge in terms of intermediate lemmata until analysis of failing proofs of the correctness theorem (and its subtheorems) provide natural suggestions for the lemmata needed to complete the proof. The suggested lemmata are handled by the same "lazy thinking paradigm" in a recursive way.

For the chosen example, the total correctness of the algorithm is proved. Termination is ensured by the fact that the splits of tuples are shorter than the tuple itself. The proof technique needed here is well-founded induction on tuples. The details of the exploration cycles will be presented during the presentation.

Literature on functional programming shows some advantages of this paradigm. As pointed out for instance in [7], functional programming offers some advantages over imperative style programming, such as being on an order of magnitude more productive (the *Theorema* system is in fact implemented in *Mathematica*, which allows programming in functional style, see [9]; one of the reasons for choosing this as the programming language was productivity), less prone to errors (no side effects at function calls), provide better modularisation. Also, as pointed out in [8], functional programs are easier to prove correct than the imperative ones. Moreover, in the case of functional programming the proofs carry over to implicitly parallel functional systems.

The *Theorema* system allows expressing functional programs and proving their correctness in the same language frame. It provides proof methods for various mathematical domains, including induction for natural numbers and sequence variables, which are applicable to proof obligations that arise in proving algorithms defined in a functional manner correct. The proofs are generated in a natural style manner, thus being easy to understand by a human reader. All these make the system a good tool for research in this area.

References

[1] B. Buchberger, *Case Study: Exploration of Sorting by Merging*, Mathematica notebooks, part of a lecture for high−school teachers, RISC-Linz, October 2001.

[2] B. Buchberger and D.Vasaru, *An Induction Prover for Equalities over Lists.* In Proceedings of the First Theorema Workshop, Hagenberg, Austria, June 9-10, 1997, RISC-Report Series No.97-20, 1997.

[3] B. Buchberger, *Mathematica as a Rewrite Language.* In: Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming), November 1-4, 1996, Shonan Village Center, (T. Ida, A. Ohori, M. Takeichieds.), World Scientific, Singapore - New Jersey - London - Hong Kong, 1996, pp. 1-13.

[4] B. Buchberger, C. Dupre, T. Jebelean, K. Kriftner, K. Nakagawa, D. Vasaru and W. Windsteiger, *The Theorema Project: A Progress Report.* In M.Kerber and M.Kolhase, editors, Proceedings of Calculemus 2000: Integration of Symbolic Computation and Mechanized Reasoning, A.K. Petersm Natick, Massatchussets, 2000.

[5] A. Craciun and B. Buchberger, *Proving the Correctness of the Merge-Sort Algorithm with Theorema*, Symbolic and Numeric Algorithms for Scientific Computing, Proceedings of the SYNASC02 Workshop, Oct. 9-12, Timisoara, Romania, Editura Mirton, Timisoara, 2002, pp. 97-111.

[6] D. Vasaru-Dupre, *Automated Theorem Proving by Integrating Proving, Solving and Computing.* The Research Institute for Symbolic Computation, Johannes Kepler University, May 2000, RISC report 00-19.

[7] J. Hughes, *Why Functional Programming Matters.* In D. Turner, editor, Research Topics in Functional Programming, Addison Wesley 1990.

[8] S. Thompson, *Proof.* In K. Hammond and G. Michaelson, editors, Research Directions in Parallel Functional Programming, pp. 93-119, Springer-Verlag, nov 1999.

[9] S. Wolfram, *The Mathematica Book.* Online version, http://documents.wolfram.com/v4.