A Practical Approach to Verification of Recursive Programs in Theorema

EXTENDED ABSTRACT

Nikolaj Popov, Tudor Jebelean*
RISC – Linz, Austria
{popov,jebelean}@risc.uni-linz.ac.at

Abstract

We report work in progress concerning the theoretical basis and the implementation in the Theorema system of a methodology for the generation of verification conditions for recursive procedures, with the aim of practical verification of recursive programs. Proving total correctness is achieved by proving separately partial correctness and then termination. We develop a pattern for proving partial correctness properties of programs which have simple functional recursive definitions, and we discuss how this can be extended to arbitrary procedural recursive programs. The method is based on Scott's Induction, from which we extract the essential features. Furthermore we develop a pattern for proving termination of simple recursive functional programs, and we discuss its extension to procedural programs.

While proving [partial] correctness of non-recursive procedural programs is quite well understood, for instance by using Hoare Logic [4], [6], there are relatively few approaches to recursive procedures (see e.g. [7] Chap. 2).

We discuss here a practical approach, based on a certain theory and including implementation, to automatic generation of verification conditions for recursive programs (both functional and procedural). The implementation is part of the *Theorema* system, and complements the

*The program verification project in the frame of e-Austria Timişoara is supported by BMBWK (Austrian Ministry of Education, Science and Culture), BMWA (Austrian Ministry of Economy and Work) and MEC (Romanian Ministry of Education and Research). The Theorema project is supported by FWF (Austrian National Science Foundation) – SFB project P1302.

work on non-recursive procedures [5, 8]. Moreover, this work is complementary to the research performed in the *Theorema* group on verification and synthesis of functional algorithms based on logic principles [1, 2].

The *Theorema* system (`www.theorema.org`, [11]) aims at realization of a computer driven assistant for the working mathematicians and engineers, which integrates automatic reasoning, algebraic computing, and equational solving. The system provides an uniform environment in natural logico-mathematical language for defining, testing, and proving properties of algorithms, and in general for creating and investigating mathematical models.

We consider the correctness problem expressed as in Hoare Logic: *given* the program (by its source text) and its specification (by a precondition on the input and a postcondition on the input and the output), *generate* the verification conditions which are [minimally] sufficient for the program to satisfy the specification. We approach the correctness problem by splitting it into two parts: *partial correctness* (prove that the program satisfies the specification provided it terminates), and *termination* (prove that the program always terminates).

Partial Correctness. For illustration we give here the formalization corresponding to a simple recursive structure. Let be the program for a function F:

$$F[X] = \textbf{ If } Q[X] \textbf{ then } S[X] \textbf{ else } T[X, F[R[X]]], \tag{1}$$

where S, Q, R, T are total functions, and let $\Phi[X], \Psi[X, Y]$ be the precondition and the postcondition of the program specification. Partial correctness means that for any input X which satisfies $\Phi[X]$, if the program terminates, then the result satisfies $\Psi[X, F[X]]$.

Using the technique known as "Scott induction" [10, 3, 9], one can prove that a sufficient condition for partial correctness is the conjunction of:

$$(\forall X : \ \Phi[X] \wedge Q[X]) \ \Psi[X, S[X]]) \tag{2}$$

$$(\forall X : \ \Phi[X] \wedge \neg Q[X]) \ \Phi[R[X]]) \tag{3}$$

$$(\forall X : \ \Phi[X] \wedge \neg Q[X]) \ (\forall Y) \ (\Psi[R[X], Y] \Longrightarrow \Psi[X, T[X, Y]]) \tag{4}$$

If the function F is intented to be total, then the formulae are simpler and (3) disappears, and if [some of] the auxiliary functions Q, R, S, T are not total (or have input preconditions), then the formulae are more complicated, but the essence remains the same: formula (2) expresses the correctness of the base case, formula (4) expresses the inductive step for the correctness of the recursive expression $T[\ldots]$ under the assumption that the reduced call $F[R[X]]$ is correct, and formula (3) expresses the appropriateness of applying F to the reduced argument $R[X]$.

Similar formulae can be constructed for more complex versions of (1), e.g. with several cases in the conditional, with more arguments to T (several R-like functions), etc. However, the verification conditions for partial correctness always express the same essence:

- Prove that the base cases are correct.
- Prove that the reduced arguments satisfy the input conditions.
- Prove that the recursive expressions are correct under the assumption that the reduced calls are correct.

This principle in fact generalizes to arbitrary functional and procedural programs, as it was noted elsewhere (e.g. [7]). The practical consequence of this theoretical basis is that (somewhat surprisingly) partial correctness of recursive procedural programs can be checked exactly in the same way as for the non-recursive programs, we do not need to make any difference between recursive and non-recursive routines. The difference only occurs when proving termination.

Termination. It is known that proving termination of a program is undecidable in general, however it is possible in many particular cases. In fact, it should be always possible to prove termination of correct programs written for practical purposes.

For programs having the structure given by (1), and provided (3) holds, termination is equivalent to termination of the function:

$$F'[X] = \ \textbf{If } Q[X] \textbf{ then } 0 \textbf{ else } F'[R[X]], \tag{5}$$

which only depends on Q and R. More exactly, in the total domain D of F (specified by Φ), Q defines a "basis" sub-domain B , and the function R must have the property that it transforms in a finite number of steps any element from D into an element from B. This condition can be expressed at object level in various ways, for instance by using a second–level predicate *"converges"*:

$$C[X] \iff (Q[X] \vee (\neg Q[X] \wedge C[R[X]])), \tag{6}$$

a general sufficient condition for termination is:

$$(\forall X : \ \Phi[X]) \ C[X] \tag{7}$$

The structure of the termination proof depends on R and Q, which typically define a certain induction principle.

The generalization of this termination conditions to more complex programming structures could be quite difficult in general, however in practical situations the programming style itself should impose the usage of such inductive domains which are simple and natural – and for these the *Theorema* system offers various inductive provers (natural numbers and tuples).

The extension of this principle to general procedural recursions is based on the idea that one should look at the reduction operations which are applied to the arguments of the main function in order to obtain the arguments of the sub-calls, and the conditions under which these occur. These define a system of conditional rewrite rules which have to converge to the "basis domain". Generating verification conditions and proofs for such problems in general is nontrivial and could be highly creative, therefore we are investigating various test cases in order to determine the typical practical situations and the most reasonable possibilities to solve them.

Implementation and experiments. Part of the methods described here are implemented in the *Theorema* system and we are studying various test cases in order to improve the power of our condition generator. Moreover, the concrete proof problems are used as test cases for our provers and for experimenting with the organization of the mathematical knowledge. During the talk we will present some of these concrete experiments.

References

[1] A. Craciun; B. Buchberger. Functional program verification with theorema. In *CAVIS-03 (Computer Aided Verification of Information Systems)*, Institute e-Austria Timisoara, February 2003.

[2] B. Buchberger. Verified algorithm development by lazy thinking. In *IMS 2003 (International Mathematica Symposium)*, Imperial College, London, July 2003.

[3] I. Soskov; A. Dichev. *A Theory of Programs.* Sofia University, 1996. In Bulgarian.

[4] C. A. R. Hoare. *An axiomatic basis for computer programming. Comm. ACM*, 12, 1969.

[5] M. Kirchner. Program verification with the mathematical software system Theorema. Technical Report 99-16, RISC-Linz, Austria, 1999. PhD Thesis.

[6] B. Buchberger; F. Lichtenberger. *Mathematics for Computer Science I - The Method of Mathematics (in German).* Springer, 2nd edition, 1981.

[7] M. C. Paull. *Algorithm Design. A recursion transformation framework.* Wiley, 1987.

[8] L. Kovacs; N. Popov. Procedural Program Verification in Theorema. In *Omega-Theorema Workshop*, 2003. Hagenberg, Austria, 25-27 May 2003.

[9] N. Popov. Operators in Recursion Theory. Technical Report 03-06, RISC-Linz, Austria, 2003.

[10] J. Loeckx; K. Sieber. *The Foundations of Program Verification.* Teubner, second edition, 1987.

[11] B. Buchberger; C. Dupre; T. Jebelean; F. Kriftner; K. Nakagawa; D. Vasaru; W. Windsteiger. Theorema: A progress report. In *Calculemus 2000 (International Symposium on Integrating Computation and deduction)*, St.Andrews, Scotland, 2000.