A Practical Approach to Proving Termination of Recursive Programs in Theorema

Nikolaj Popov, Tudor Jebelean*

Research Institute for Symbolic Computation, Linz, A–4232 Hagenberg, Austria
popov@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. In contrast to other approaches, which use a special theory describing the behavior of programs, we use such a theory only "in the background", for developing a general rule for generating verification conditions, while the conditions themselves are presented (and provable) using the theories relevant to the program text only. This is very important for automatic proving, since it reduces significantly the effort of the provers. We performed practical experiments in which various programs are completely verified using the verification condition generator and the provers of the Theorema system.

Introduction. While proving [partial] correctness of non-recursive procedural programs is quite well understood, for instance by using Hoare Logic [3], [6], there are relatively few approaches to recursive procedures (see e.g. [8] Chap. 2).

We discuss here a practical approach to automatic generation of verification conditions for functional recursive programs, partially based on Scott induction in the fixpoint theory of programs [13,11,7,9] and the implementation of this approach. The implementation is part of the *Theorema* system, and complements the research performed in the *Theorema* group on verification and synthesis of functional algorithms based on logic principles [1,2,4].

The *Theorema* system (`www.theorema.org`, [12]) aims at realization of a computer aided 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 follows: *given* the program (by its source text) which computes the function F and given its specification by a precondition on the input $I_F[x]$ and a postcondition on the input and the output $O_F[x, y]$, *generate* the verification conditions which are [minimally] sufficient for the program to satisfy the specification.

For simplifying the presentation, we consider here the "homogeneous" case: all functions and predicates are interpreted over the same domain. Proving the verification conditions

* 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. Additional support comes from the EU project CALCULEMUS (HPRN-CT-2000-00102).

will require the *specific theory* relevant to this domain and to the auxiliary functions and predicates which occur in the program.

The functional program of F can be interpreted as a set of predicate logic formulae, and the correctness of the program can be expressed as:

$$\forall x \; I_F[x] : O_F[x, F[x]], \tag{1}$$

which we will call the *correctness formula* of F. In order for the program to be correct, the correctness formula (1) must be a logical consequence the formulae corresponding to the definition of the function (and the specific theory). This approach was previously used by other authors and is also experimented in the Theorema system [1]. However, the proof of such one-single theorem may be difficult, because the prover has to find the appropriate induction principle and has to find out how to use the properties of the auxiliary functions present in the program.

The method presented in this paper generates several verification conditions, which are easier to prove. In particular, only the termination condition needs an inductive proof, and this termination condition is "reusable", because it basically expresses an induction principle which may be useful for several programs. This is important for automatic verification embedded in a practical verification system, because it leads to early detection of bugs (when proofs of simpler verification conditions fail).

Moreover, the verification conditions are provable in the frame of predicate logic, without using any theoretical model for program semantics or program execution, but only using the theories relevant to the predicates and functions present in the program text. This is again important for the automatic verification, because any additional theory present in the system will significantly increase the proving effort.

We start by developing a set of rules for generating verification conditions, for programs having a particular structure. The rules for partial correctness are developed using Scott induction and the fixpoint theory of programs, however the verification conditions themselves do not refer to this theory, they only state facts about the predicates and functions present in the program text. In particular, the termination condition consists in a property of a certain simplified version of the original program. By inspecting the shape of these rules for several program structures, it is possible to derive a more general rule for the derivation of verification conditions, such that the correctness formula (see above) **is a logical consequence of these verification conditions** in the frame of predicate logic, without using any model of computation.

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). Proving *partial correctness* may be achieved by Scott's induction [13,11,7,9] – a detailed description of the method for a certain class of functional programs is presented in [10].

Example: Simple Recursive Programs. Let be the program:

$$F[x] = \textbf{ If } Q[x] \textbf{ then } S[x] \textbf{ else } C[x, F[R[x]]],$$

where Q is a predicate and S, C, and R are auxiliary functions whose total correctness is assumed (S is a "simple" function, C is a "combinator" function, and R is a "reduction" function). Note that the program above can be seen as an abbreviated notation for the logical formulae:

$$\forall x\ (Q[x] \implies F[x] = S[x])$$

$$\forall x\ (\neg Q[x] \implies F[x] = C[x, F[R[x]]]).$$

Using Scott induction in the fixpoint theory of functions, one obtains the following verification conditions for the *partial correctness* of the program of F :

$$\forall x\ I_F[x] :\ (Q[x] \implies O_F[x, S[x]])$$

$$\forall x, y\ I_F[x] :\ ((\neg Q[x] \wedge O_F[R[x], y]) \implies O_F[x, C[x, y]])$$

(In fact, the conditions can be further decomposed using the preconditions and the post-conditions of the auxiliary functions, see [5].)

The condition for the *termination* of the program can be expressed using a simplified version of the initial function:

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

which only depends on Q and R. Namely, the verification condition is

$$\forall x\ I_F[x] : F'[x] = 0, \tag{2}$$

which must be proven based on the logical formulae corresponding to the definition of F' (and the local theory relevant to the program). The condition follows from the equivalence of the termination properties of F and F', which can be proven in the fixpoint theory of functions e. g. by using induction on the number of recursion steps (see [14]).

The termination condition can be further refined to:

$$\forall x\ I_F[x] : ((Q[x] \Longrightarrow P[x]) \wedge (\neg Q[x] \Longrightarrow (P[R[x]] \Longrightarrow P[x])) \implies \forall x P[x]),$$

where P is a new predicate symbol, which is a proof in second order logic of the formula above for any P. This condition defines in fact an induction principle. By taking $P[x] \Longleftrightarrow (F'[x] = 0)$, one can obtain a proof of (2). By taking $P[x] \Longleftrightarrow O_F[x, F[x]]$, one can obtain a proof of the correctness formula (1).

Note that both conditions only depend on Q and R, thus they abstract some part of the function definition. In practical programming, these Q and R correspond to typical algorithm schemes (take e. g. $Q[x] \Longleftrightarrow (x = 0)$ and $R[x] = (x - 1)$), thus the termination condition (whose proof usually involves induction) is usable for several programs.

Generalization. By applying the same method to other recursion schemes, one notes that the verification conditions for the partial correctness can be generated directly (without using an additional theory) by applying few basic principles:

- check the input conditions when calling subroutines;
- accumulate all reasonable assumptions (coming from the input condition of the main function, from **if–then–else** statements and from the correctness formulae of the subroutines),
- try to finally obtain the correctness property for the output of F.

Furthermore, the termination condition can be generated either:

- as the "identical zero" property of the simplified function, or
- as the appropriate induction principle which makes the correctness formula a logical consequence the partial correctness conditions.

Implementation and experiments. The methods described above are implemented in the *Theorema* system and we are studying various recursive schemes and several test cases in order to improve the power of the verification condition generator. Furthermore, the concrete proof problems are used as test cases for the provers of *Theorema* and for experimenting with the organization and management of the mathematical knowledge. Currently we are able to generate the verification conditions and to prove them automatically in the *Theorema* system for many concrete programs.

References

1. A. Craciun, B. Buchberger. Functional Program Verification with Theorema. In *Computer Aided Verification of Information Systems (CAVIS-04)*, 2003. Technical Report 03-05, Institute e-Austria Timisoara (`www.ieat.ro`).
2. B. Buchberger. Verified Algorithm Development by Lazy Thinking. In *International Mathematica Symposium (IMS 2003)*, Imperial College, London, July 2003.
3. C. A. R. Hoare. *An axiomatic basis for computer programming. Comm. ACM*, 12, 1969.
4. T. Jebelean, L. Kovacs, N. Popov. Verification of Imperative Programs in Theorema. In *1st South-East European Workshop in Formal Methods (SEEFM03)*, Thessaloniki, Greece, 20 November 2003.
5. T. Jebelean. Forward Verification of Recursive Programs. In *Computer Aided Verification of Information Systems (CAVIS-04)*, 2004. Technical report 04-01, Institute e-Austria Timisoara (`www.ieat.ro`).
6. B. Buchberger, F. Lichtenberger. *Mathematics for Computer Science I - The Method of Mathematics. (in German)* Springer, 2nd edition, 1981.
7. Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill Inc., 1974.
8. M. C. Paull. *Algorithm Design. A recursion transformation framework.* Wiley, 1987.
9. N. Popov. Operators in Recursion Theory. Technical Report 03-06, RISC-Linz, Austria, 2003.
10. N. Popov, T. Jebelean. A Practical Approach to Verification of Recursive Programs in Theorema. In T. Jebelean and V. Negru, editors, *Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003)*, Timisoara, Romania, 1–4 October 2003.
11. J. Loeckx, K. Sieber. *The Foundations of Program Verification.* Teubner, second edition, 1987.
12. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. Theorema: A progress report. In *International Symposium on Integrating Computation and deduction (Calculemus 2000)*, St.Andrews, Scotland, 2000.
13. J. W. de Bakker, D. Scott. A Theory of Programs. In *IBM Seminar*, Vienna, Austria, 1969.
14. N. Popov, T. Jebelean. Verification of Simple Recursive Programs: Sufficient Conditions. Technical Report 04-06, RISC-Linz, Austria, 2004.