Verification of Simple Recursive Programs in Theorema: Sufficient Conditions

Nikolaj Popov and Tudor Jebelean [⋆]

Research Institute for Symbolic Computation,
University of Linz, 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. We develop a method for proving total correctness properties of programs which have simple functional recursive definitions, and we discuss its different aspects. Most of the verification conditions are expressed in first order logic and their proof does not need a theory of computation, but only the knowledge which is specific to the functions occuring in the program.

Introduction

We discuss here a practical approach to automatic generation of verification conditions for functional recursive programs. 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 [Buc03a,Buc03b,Pop03].

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 and a postcondition on the input and the output O_F, *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 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. By correctness of the program we mean both *partial correctness* (1) and *termination* (2):

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge F[x] \downarrow \Longrightarrow O_F[x, F[x]]), \tag{1}$$

[⋆] 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.

$$(\forall x \in \mathbb{D})\ (I_F[x] \Longrightarrow F[x] \downarrow), \tag{2}$$

where the $\downarrow$ is the predicate expressing termination. As one sees, (3) is a logical consequence of (1) and (2), but not the vice versa (for example I_F and O_F are *true* but F never terminates, i.e. $F[x] \downarrow$ is *false* for any x.

$$(\forall x \in \mathbb{D})\ (I_F[x] \Longrightarrow O_F[x, F[x]]), \tag{3}$$

In order for the program to be correct, the correctness formula (3) must be a logical consequence of the formulae corresponding to the definition of the function (and the specific theory). Additionally, one needs to ensure termination, which we study for a certain class of problems.

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 termination property of a certain simplified version of the original program.

We consider programs, defined in a domain $\mathbb{D}$, like naturals $\mathbb{N}$, reals $\mathbb{R}$ etc. For the purpose of our study, we extend the domain $\mathbb{D}$ to a new domain $\mathbb{D} \cup \{\perp\}$, where $\perp$ is an additional symbol for expressing the nonterminating constant. In the extended domain the programs F are functions from $\mathbb{D} \cup \{\perp\} \rightarrow \mathbb{D} \cup \{\perp\}$. The predicates Q which are used in the programs, become functions from $\mathbb{D} \cup \{\perp\} \rightarrow \{\mathbb{T}, \mathbb{F}\} \cup \{\perp\}$. The unary predicate $\downarrow$ is defined by the formula: $(\forall n)(n \downarrow \Longleftrightarrow n \neq \perp)$ Here we make the natural assumption: For any function F $F[\perp] = \perp$.

In addition we will need the nowhere defined function Ω which is defined as $(\forall d)(\Omega[d] = \perp)$.

We say that a function f is total with respect to a precondition I if and only if $(\forall d \in \mathbb{D})(I[d] \Longrightarrow f[d] \downarrow)$

We will use the restricted graph function defined as

$$(\forall f)(RGraph[f] = \{\langle x, y\rangle : f[x] = y \wedge y \downarrow\}).$$

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 induction [dBDS69], [Sie87], [Man74], [Sto77], [Gun92].

Simple Recursive Programs

We define here a class of recursive programs, defined by the program scheme (4). A program defined by (4) is called simple recursive program.

Given a domain $\mathbb{D}$, let (4) be the program for a function F:

$$(\forall x \in \mathbb{D})\ F[x] = \ \textbf{If}\ Q[x]\ \textbf{then}\ S[x]\ \textbf{else}\ C[x, F[R[x]]], \tag{4}$$

where Q is a predicate on $\mathbb{D}$ and S, C, and R are auxiliary functions (S is a "simple" function whose definition does not use F, C is a "combinator" function, and R is a "reduction" function).

We consider only the scheme when the functions F, S and R and the predicate Q are of arity 1 and the function C of arity 2. However, if $\mathbb{D}$ is a vector domain, then different arities will be treated by the same method (see the example of *rem* (30)).

Considering the fixpoint theory [Man74,Sto77,Gun92], we look at program definitions as definitions of operators. The programs are functions, defined by the least fixpoint of the corresponding operators. In our case, the operator Γ corresponding to (4), is

$$\lambda F.\lambda x.\ \textbf{If}\ Q[x]\ \textbf{then}\ S[x]\ \textbf{else}\ C[x, F[R[x]]], \tag{5}$$

The finite approximations of Γ are defined recursively as follows:

$$F_0 = \Omega$$

$$F_n = \Gamma[F_{n-1}].$$

The least fixpoint F_Γ of Γ is defined as the union of the finite approximations

$$F_\Gamma = \bigcup_n F_n. \tag{6}$$

From this prospective, it is easy to see how Scott induction principle works. Let ϕ be a property on functions such that

– $\phi[\Omega]$ holds;

– $\phi[f] \implies \phi[\Gamma[f]]$ holds for any f;

then we can infer $\phi[F_\Gamma]$.

As it is known, not every property ϕ is admissible. However, properties which express partial correctness (7) are known to be admissible. A property ϕ is said to be partial Correctness property if and only if there are predicates I and O, such that:

$$(\forall f)(\phi[f] \iff (\forall a)(f[a]\downarrow \wedge I[a] \Longrightarrow O[a, f[a]])). \tag{7}$$

Partial Correctness

Here we present the Partial Correctness Verification Conditions and we prove their sufficiency for the program (4) to be partially correct.

Now, we make several assumptions which we will use till the end of this paper. Given a domain $\mathbb{D}$, let (4) be a simple recursive program computing the function F. Let $I_F[x], O_F[x, y]$ be the precondition and the postcondition of F. Let $I_S[x], O_S[x, y], I_C[x, y], O_C[x, y, z], I_R[x], O_R[x, y]$ be the preconditions and the postconditions of the auxiliary functions. Assume that the correctness formula holds for each of them, i.e., the total correctness of the auxiliary functions is assumed. In this case, the following statement holds:

Lemma 1. *If the following three formulae:*

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge Q[x] \wedge S[x] \downarrow \Longrightarrow O_F[x, S[x]]) \tag{8}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \wedge R[x] \downarrow \Longrightarrow I_F[R[x]]) \tag{9}$$

$$\begin{aligned}(\forall x \in \mathbb{D})\ (I_F[x] \wedge C[x, F[R[x]]] \downarrow \wedge \neg Q[x] \wedge O_F[R[x], F[R[x]]] \ \Longrightarrow \\ O_F[x, C[x, F[R[x]]]])\end{aligned} \tag{10}$$

are valid, then the program (4) *is partially correct, that is* (1) *holds.*

Proof. Assume that (8), (9) and (1) hold. Consider the following property:

$$(\forall f)(\phi[f] \iff (\forall a)(f[a] \downarrow \wedge I_F[a] \Longrightarrow O_F[a, f[a]])).$$

It is a partial correctness property and we apply Scott induction for inferring $\phi[F]$.

–$\phi[\Omega]$ holds.

–Assume $\phi[f]$ for some f, that is

$$(\forall a)(f[a] \downarrow \wedge I_F[a] \Longrightarrow O_F[a, f[a]]). \tag{11}$$

Show $\phi[f']$, where $f' =$ **If** $Q[x]$ **then** $S[x]$ **else** $C[x, F[R[x]]]$, that is

$$(\forall a)(f'[a] \downarrow \wedge I_F[a] \Longrightarrow O_F[a, f'[a]]). \tag{12}$$

Take arbitrary but fixed a and assume that $f'[a] \downarrow$ and $I_F[a]$.

a)case: $Q[a]$. So, we have $f'[a] = S[a]$. From $f'[a] \downarrow$, we obtain that $S[a] \downarrow$. From here, by (8) we obtain $O_F[a, S[a]])$ and so $O_F[a, f'[a]])$.

b)case: $\neg Q[a]$. So, we have $f'[a] = C[a, f[R[a]]]$. From $f'[a] \downarrow$, we obtain that

$$C[a, f[R[a]]] \downarrow \tag{13}$$

$$f[R[a]] \downarrow \tag{14}$$

$$R[a] \downarrow . \tag{15}$$

From $I_F[a]$, by (9), we obtain $I_F[R[a]]$. From here and (14), by (11) we obtain that $O_F[R[a], f[R[a]]]$. From here, by (1), we obtain $O_F[a, C[a, f[R[a]]]]$ and hence $O_F[a, f'[a]]$.

From here we infer that the property ϕ holds for the least fixpoint F, hence (1) holds.

Termination

Here we present the Termination Verification Conditions and we prove their sufficiency for the program (4) to terminate. In addition to the assumptions from the beginning of the previous section, we assume that (8), (9) and (1) hold.

The following lemma expresses that for an input a, if there exists a finite number n of applications of R to a, such that Q holds then F terminates on a.

Lemma 2. *If the following three formulae:*

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge Q[x] \implies I_S[x]) \tag{16}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \implies I_R[x]) \tag{17}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \wedge O_F[R[x], F[R[x]]] \implies I_C[x, F[R[x]]]) \tag{18}$$

hold, then

$$(\forall x \in \mathbb{D})\ (I_F[x] \implies (\exists n \in \mathbb{N})(Q[R^n[x]] \implies F[x]\downarrow))$$

holds as well.

Proof. Assume that (16), (17) and (18) hold. Assume, also $a \in \mathbb{D}$, $I_F[a]$ and $Q[R^n[a]]$ for some $n \in \mathbb{N}$.

1)case: $n = 0$. This means that $Q[a]$ and hence $F[a] = S[a]$. From here and (16) by knowing that S is totally correct, we obtain that $S[a]\downarrow$, which implies $F'[a]\downarrow$.

2) case: $n > 0$. Assume that n is the first such that $Q[R^n[a]]$, that is $\neg Q[R^i[a]]$ for any $0 \leq i < n$.
Using induction on i, we will show that $F[R^{n-i}[a]]\downarrow$ and $O_F[R^{n-i}[a], F[R^{n-i}[a]]]$.

a) Base case: $i = 0$.

$F[R^{n-i}[a]] = F[R^n[a]] = S[a]$, hence $F[R^{n-i}[a]]\downarrow$.

From $I_F[a]$ and $\neg Q[R^i[a]]$ for any $0 \leq i < n$, by (9) we obtain $I_F[R[a]]$, ... $I_F[R^n[a]]$.

From $I_F[R^n[a]]$ and $Q[R^n[a]]$ by (16) we obtain $I_S[R^n[a]]$. From here, by knowing that S is totally correct, we obtain that $S[R^n[a]]\downarrow$, which implies $F[R^{n-0}[a]]\downarrow$.

From here and $Q[R^n[a]]$, by (8) we obtain $O_F[R^n[a], F[R^n[a]]]$.

b) Induction step: $i > 0$.

Assume that $F[R^{n-i}[a]]\downarrow$ and $O_F[R^{n-i}[a], F[R^{n-i}[a]]]$.

$F[R^{n-(i+1)}[a]] = F[R^{n-i-1}[a]] = C[R^{n-i-1)}[a], F[R[R^{n-i-1}[a]]]] =$
$= C[R^{n-i-1)}[a], F[R^{n-i}[a]]]$

From the induction hypothesis, by (18) we obtain $I_C[R^{n-i-1)}[a], F[R^{n-i}[a]]]$.
From here, by knowing that C is totally correct, we obtain that
$C[R^{n-i-1)}[a], F[R^{n-i}[a]]]\downarrow$, which implies $F[R^{n-i-1}[a]]\downarrow$.

From here, by (8) we obtain $O_F[R^{n-i-1}[a], F[R^{n-i-1}[a]]]$, which completes the proof of the lemma.

Here we define a new program F' and we call it "Simplified version of F".

$$(\forall x \in \mathbb{D})\ F'[x] = \textbf{ If } Q[x] \textbf{ then } 0 \textbf{ else } F[R[x]] \tag{19}$$

The following lemma expresses that the termination of the simplified version F' on an input a, ensures that there exists a finite number n of applications of R to a, such that Q holds.

Lemma 3.

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge F'[x] \downarrow \Longrightarrow (\exists n \in \mathbb{N})(Q[R^n[x]])).$$

Proof. By contradiction.

Take arbitrary but fixed a and assume $I_F[a]$.

Assume $F'[a] \downarrow$ and $(\forall n \in \mathbb{N})(\neg Q[R^n[a]])$.

From here we obtain that $F'[a] \neq \perp$ and hence $\exists b \in \mathbb{D})(\langle a, b\rangle \in RGraph[F'])$.

Let f_0, f_1, ... f_m are the finite approximations of F', i.e., $F' = \bigcup_i f_i$.

From $\langle a, b\rangle \in RGraph[F'] = RGraph[\bigcup_i f_i]$ follows
$(\exists k > 0)(\langle a, b\rangle \in RGraph[f_k])$ ($k > 0$ because $a, b \neq \perp$). From here, we obtain that $f_k[a] = b$ which implies $f_k[a] \neq \perp$. By the definition
$f'_k[a] = \textbf{ If } Q[a] \textbf{ then } 0 \textbf{ else } f_{k-1}[R[a]]$. From $(\neg Q[[a]])$ (it is so, because we have $(\forall n \in \mathbb{N})(\neg Q[R^n[a]]))$, we obtain $f_k[a] = f_{k-1}[R[a]]$. Since $f_k[a] \downarrow$, we have $f_{k-1}[R[a]] \downarrow$. Applying the same reason k times, we obtain that $f_k[a] = f_{k-k}[R^k[a]]$ and hence $f_{k-k}[R^k[a]] \downarrow$. This contradicts to the definition of $f_0 = \Omega$.

The next statement gives the Termination Verification Conditions. It basically expresses that if all the calls to the auxiliary functions obey their input conditions and in addition the simplified version terminates, then the main program terminates as well.

Lemma 4. *If* (16)*,* (17)*,* (18) *and*

$$(\forall x \in \mathbb{D})\ (I_F[x] \implies F'[x] \downarrow) \tag{20}$$

are valid, then the program (4) *terminates, that is* (2) *holds.*

Proof. Assume that (16), (17), (18) and (20) hold. Take arbitrary but fixed a and assume $I_F[a]$.

a)case: $Q[a]$. So, we have $F[a] = S[a]$. From (16) follows $I_S[a]$, hence $S[a] \downarrow$ and $F[a] \downarrow$.

b)case: $\neg Q[a]$. So, we have $F[a] = C[a, F[R[a]]]$. It suffices to prove that

(1) $R[a] \downarrow$

(2) $F[R[a]] \downarrow$

(3) $C[a, F[R[a]]] \downarrow$

Prove (1): From $I_F[a]$ and $\neg Q[a]$ by (17), we obtain $R[a] \downarrow$.

Prove (2): It follows by Lemma 2.

Prove (3): From $I_F[a]$ and $\neg Q[a]$ by (9), we obtain $I_F[R[a]]$. From this and $F[R[a]] \downarrow$ by the partial correctness of F, we obtain $O_F[R[a], F[R[a]]]$. From here, by (18), we obtain $C[a, F[R[a]]] \downarrow$.

Total Correctness

Here we present the Total Correctness Verification Conditions which are obtained as a combination of the Partial Correctness and the Termination Verification Conditions. This a "Soundness" theorem, because it ensures that the correctness of the Verification Conditions implies the correctness of the program.

Theorem 1. *If the following seven formulae:*

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge Q[x] \implies O_F[x, S[x]]) \tag{21}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \implies I_F[R[x]]) \tag{22}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \wedge O_F[R[x], F[R[x]]] \implies O_F[x, C[x, F[R[x]]]]) \tag{23}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge Q[x] \implies I_S[x]) \tag{24}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \implies I_R[x]) \tag{25}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \wedge \neg Q[x] \wedge O_F[R[x], F[R[x]]] \implies I_C[x, F[R[x]]]) \tag{26}$$

$$(\forall x \in \mathbb{D})\ (I_F[x] \implies F'[x]\downarrow) \tag{27}$$

are valid, then the program (4) *is totally correct, that is* (3) *and* (2) *hold.*

Proof. Assume that (21), (22), (23), (24), (25), (26) and (27) hold. First we show termination. From here, by Lemma 4 we obtain that (2) holds.

Now we show partial correctness. From (21) follows (8). From (22) follows (9). From (23) follows (1). From here, by Lemma 1 we obtain (1) and by (2) obtain (3), which completes the proof of the theorem.

As one sees, all the verification conditions without the last one, are first order predicate logic formulae. The proof of their validity would involve only the theory of the domain $\mathbb{D}$, without any additional knowledge concerning *computation*, $\perp$ etc.

An alternative to (27) is the following verification condition:

$$(\forall x \in \mathbb{D})\ (I_F[x] \implies (\exists n \in \mathbb{N})(Q[R^n[x]])). \tag{28}$$

However, we prefer to keep (27), because it is more intuitive. Moreover, it might be used in the following manner: We form a set of F'–like functions whose termination is proven (or assumed as axioms). Then, whenever we need to prove (27), we first check for membership to this set. If yes, then we are done. If not, we try to prove it separately. Within this proof, we may replace (27) by its alternative form (28), and prove it as first order predicate logic formula. Once we manage, we add it to the set.

The so formed set plays a very important from the automatic theorem proving point of view role, because its elements are reusable. For example, (29) is the *simplified version* of all the primitive recursive programs on one argument.

$$Prim[x] = \textbf{ If } x = 0 \textbf{ then } 0 \textbf{ else } Prim[x-1], \tag{29}$$

with: $\mathbb{D} = \mathbb{N}$ and $I_{Prim}[x] \iff \mathbb{T}$.

Examples

For better understanding, we give here an example of a program and its specification and we show the verification conditions.

Consider the following program:

$$rem[x,y] = \textbf{ If } x < y \textbf{ then } x \textbf{ else } rem[x-y,y] \tag{30}$$

The specification of *rem* is: The domain $\mathbb{D} = \mathbb{N}^2$, the precondition

$$I_{rem}[x,y] \iff y > 0$$

and the postcondition

$$O_{rem}[x,y,z] \iff (\exists q \in \mathbb{N})(x = z + y * q \wedge z < y).$$

The (automatically) generated verification conditions for the *total correctness* of the function *rem* are:

$$\begin{aligned}
&(\forall x, y \in \mathbb{N})\ (y > 0 \wedge x < y \implies (\exists q \in \mathbb{N})(x = x + y * q \wedge x < y))\\
&(\forall x, y \in \mathbb{N})\ (y > 0 \wedge x \geq y \implies (y > 0)\\
&(\forall x, y, b \in \mathbb{N})\ (y > 0 \wedge x \geq y \wedge (\exists q \in \mathbb{N})(x - y = b + y * q \wedge b < y) \implies\\
&\qquad (\exists q \in \mathbb{N})(x = b + y * q \wedge b < y))\\
&(\forall x, y \in \mathbb{N})\ (y > 0 \wedge x < y \implies \mathbb{T})\\
&(\forall x, y \in \mathbb{N})\ (y > 0 \wedge x \geq y \implies \mathbb{T})\\
&(\forall x, y \in \mathbb{N})\ (y > 0 \wedge x \geq y \wedge (\exists q \in \mathbb{N})(x - y = b + y * q \wedge b < y) \implies \mathbb{T})\\
&(\forall x, y \in \mathbb{N})(y > 0 \implies F'[x,y] \downarrow,
\end{aligned}$$

where

$$F'[x,y] = \textbf{ If } x < y \textbf{ then } 0 \textbf{ else } F'[x-y,y].$$

Here the symbol $\mathbb{T}$ stands for the logical constant *true*. In the example the preconditions of all the auxiliary functions are $\mathbb{T}$.

If we manage to proof the verification conditions, then we are sure that the program meets its specification.

Note that, although the functions *rem* and $-$ and the predicate $<$ have arity 2, the program scheme is still (4), by taking $\mathbb{D} = \mathbb{N}^2$.

Conclusions and Further Work

The problem of verifying recursive programs having certain shape we transfer into a problem of proving first order predicate logic formulae by generating verification conditions.

Starting from the most elementary recursive scheme (simple recursion (4)), we are investigating in more general schemes in our ongoing work. Among the immediately coming steps, one needs to mention:

– Recursive schemes having multiple recursive calls;
– Recursive schemes having more then one *else* branch;
– Corecursive schemes.

Another direction we are investigating, is proving the generated verification conditions by the *Theorema* system in a fully automatic manner. Moreover, the concrete proof problems are used as test cases for our provers and for experimenting with the organization of the mathematical knowledge.

References

[Buc03a] A. Craciun; B. Buchberger. Functional Program Verification with Theorema. In *CAVIS-03 (Computer Aided Verification of Information Systems)*, Institute e-Austria Timisoara, February 2003.

[Buc03b] B. Buchberger. Verified Algorithm Development by Lazy Thinking. In *IMS 2003 (International Mathematica Symposium)*, Imperial College, London, July 2003.

[dBDS69] J. W. de Bakker; D. Scott. A Theory of Programs. In *IBM Seminar*, 1969. Vienna, Austria, 1969.

[Gun92] C. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992.

[Man74] Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill Inc., 1974.

[Pop03] T. Jebelean; L. Kovacs; N. Popov. Verification of Imperative Programs in Theorema. In *1st South-East European Workshop in Formal Methods (SEEFM03)*, 2003. Thessaloniki, Greece, 20 November 2003.

[Sie87] J. Loeckx; K. Sieber. *The Foundations of Program Verification.* Teubner, second edition, 1987.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach.* MIT Press, 1977.