C Program Verification: Verification Condition Explanation and Standard Library

A. V. Promsky
A.P. Ershov Institute of Informatic Systems, Siberian Branch, Russian Academy of Sciences, Novosibirsk, Russia
e-mail: promsky@iis.nsk.su
Received September 27, 2011

Abstract—Two lines of developing the C program verification project at the A.P. Ershov Institute of Informatic Systems are presented. Firstly, the axiomatic semantics of the C-kernel language was extended by the semantic labelling. The labels introduced in the Hoare calculus correspond to various concepts inherent in verification conditions (VC). These labels can be extracted from terms and rendered into explanations written in the natural language. User friendly explanations can play a crucial role in VC understanding and error localization. Secondly, a subset of the C standard library was specified. The specifications written in ACSL correspond to the C-light memory model. Examples illustrating the use of the proposed techniques are presented.

Keywords: verification, specification, axiomatic semantics, C-light language, ACSL, verification condition, standard library
DOI: 10.3103/S0146411612070127

1. INTRODUCTION

The verification project of C-programs has been successfully developed in the Theoretical Programming Laboratory of the Institute of Informatic Systems of the Siberian Branch of the Russian Academy of Sciences [7–9]. The input language C-light is a substantial subset of the standard C99. Its formal definition was specified in the form of the structural operational semantics. The bounded kernel that is the C-kernel language with sound axiomatic semantics was selected within C-light. The verification process involves two steps. Firstly, the annotated program in C-light is translated into an equivalent C-kernel program. Secondly, the verification conditions are derived by means of the axiomatic semantics.

The successful development of the theoretical fundamentals of the project allows one to proceed to the solution of practical problems. One of them involves the localization of errors and the interpretation of the verification conditions. The axiomatic Hoare method generally correlates the conditions with the linear sections in the original program. The falseness of the condition signalizes the potential errors on the corresponding section (or in its annotations) without providing their accurate location. The situation is also difficult due to the complexity of the conditions for languages such as C, which hinders the hand held analysis.

To solve this problem, the approach presented in [3] was adapted. The idea is in the systematic enhancement of the Hoare rules so that the calculus itself can derive explanations for the conditions and correlate the condition fragments with the separate operations within a program.

The other practical problem lies in the absence of formal specifications for the standard library of the C language. The specification language ACSL [1] seems to be a suitable tool for this problem. Specifications for a series of library types and functions, which relate to the file I/O, work with memory, string manipulations, and mathematical functions, are proposed in the present article.

The paper consists of seven sections. Section 2 gives a brief description of the C-kernel language. The semantic labeling and label transformation are considered in section 3. Section 4 discusses the specification of the standard library. Section 5 gives examples of these approaches, and section 6 is dedicated to a review of similar papers. The obtained results and plans for the future are presented in the conclusions.
2. C-KERNEL

The C-kernel language significantly limits the input language C-light. Let us give a brief description of C-light [7] as compared with the standard C99. Firstly, a small number of constructions with the semantics depending on the low-level implementation details (bit fields and unions) are not maintained. Secondly, we state a series of behavior aspects that are not defined by the standard itself. For example, the expressions are computed from the left to right, and the side effects immediately come into action.

Declarations. Lists of declarators in C-kernel are allowed in the function declarations only. The storage specifiers are given in an explicit form.

Expressions. The main purpose is that any expression differing from the function call has no more than one side effect. Therefore, complex assignments and the ‘comma’ operator are prohibited in C-kernel. We traditionally use the operations new and delete in the C++ style to work with the memory.1

The normalized expressions (N Exp) make up the base in C-kernel. They are built by the standard rules of the C language with using the primary, unary, and binary operations and type casts. Thus, no memory change can take place in a normalized expression.

Let CV denote the names of the constants and variables. Then, the syntax of the expressions of C-kernel can be described as

\[
Exp ::= f(CV, ..., CV) \mid NExp = f(CV, ..., CV) \mid NExp = NExp \mid NExp = \text{new } \text{Type} \{NExp\}_\text{opt} \mid \text{delete} \{NExp\}_\text{opt}.
\]

Statements. The C-kernel language admits the following statements:

\[
Stat ::= Exp; \mid l: Stat; \mid \text{goto} \ l; \mid \text{return} \ NExp_{\text{opt}}; \mid \text{if} \ (NExp) \ Stat \ \\
\text{else} \ Stat \mid \text{while} \ (NExp) \ Stat \mid \{Stat \ldots Stat\}.
\]

Although C-kernel is a very restricted subset, any C-light program can be translated into an equivalent one in C-kernel. Table 1 gives an example of such a translation.

---

1 It is for simplifying semantics, but modern specified libraries makes it possible not to use artificial elements.