Theoretical Computer Science 631 (2016) 73–96

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Completeness for recursive procedures in separation logic

Mahmudul Faisal Al Ameen [a], Makoto Tatsuta [b,*]

[a] Department of Informatics, SOKENDAI (The Graduate University for Advanced Studies), 2-1-2 Hitotsubashi, 101-8430 Tokyo, Japan
[b] National Institute of Informatics, 2-1-2 Hitotsubashi, 101-8430 Tokyo, Japan

ARTICLE INFO

Article history:
Received 20 April 2015
Received in revised form 26 February 2016
Accepted 2 April 2016
Available online 11 April 2016
Communicated by D. Sannella

Keywords:
Separation logic
Hoare's logic
Completeness
Recursive procedures

ABSTRACT

This paper proves the completeness of an extension of Hoare's logic and separation logic for pointer programs with mutual recursive procedures. This paper shows the expressiveness of the assertion language as well. This paper achieves a new system by introducing two new inference rules, and removes an axiom that is unsound in separation logic and other redundant inference rules for showing completeness. It introduces a novel expression that is used to describe complete information of a given state in a precondition. This work also uses the necessary and sufficient precondition of a program for the abort-free execution, which enables us to utilize strongest postconditions.

1. Introduction

It is widely accepted that a program is needed to be verified to ensure that it is correct. A correct program guarantees to perform the given task as expected. It is very important to ensure the safety of the mission-critical, medical, spacecraft, nuclear reactor, financial, genetic-engineering and simulator programs. Moreover, everyone desires bug-free programs.

Formal verification is intended to deliver programs which are completely free of bugs or defects. It verifies the source code of a program statically. So formal verification does not depend on the execution of a program. The time required to verify a program depends on neither its runtime and memory complexity nor the magnitude of its inputs. A program is required to be verified only once since it does not depend on test cases. Hence, formal verification of programs is important to save both time and expenses commercially and for its supremacy theoretically. Among formal verification approaches, model checking and Hoare's logic are prominent. Model checking computes whether a model satisfies a given specification, whereas Hoare's logic shows it for all models by provability [9].

Since it was proposed by Hoare [8], numerous works on Hoare's logic have been done [1,6,11,4,7]. Several extensions have also been proposed [1], among which some attempted to verify programs that access heap or shared resources. But until the twenty-first century begins, very few of them were simple enough to use. On the other hand, since the development of programming languages like C and C++, the usage of pointers in programs (which are called pointer programs) gained much popularity for their ability to use shared memory and other resources directly and for faster execution. Yet this ability also causes crashed programs for some reasons because it is difficult to keep track of each memory operation. It may lead to unsafe heap operation. A program crash occurs when the program tries to access a memory cell that has already been deallocated before or when a memory cell is accessed before its allocation. So apparently it became necessary to have an extension of Hoare's logic that can verify such pointer programs. In 2002, Reynolds proposed separation logic [15]. It was a

* Corresponding author.
E-mail addresses: alameen@nii.ac.jp (M. Faisal Al Ameen), tatsuta@nii.ac.jp (M. Tatsuta).

http://dx.doi.org/10.1016/j.tcs.2016.04.004
0304-3975/

breakthrough to achieve the ability to verify pointer programs. Especially it can guarantee safe heap operations of programs. Although recently we can find several works on separation logic [2,10] and its extensions and applications [12,13,3], there are few works found to show their completeness [16]. Tatsuta et al. [16] show the completeness of the separation logic for pointer programs which is introduced in [15]. In this paper, we will show the completeness of an extended logical system. Our logical system is intended to verify pointer programs with mutual recursive procedures. Among several versions of the same inference rule Reynolds offered in [15] for separation logic, a concise set of backward reasoning rules has been chosen in [16]. The later work in [16] also offers rigorous mathematical discussions. The problems regarding the completeness of Hoare's logic, the concept of relative completeness, completeness of Hoare's logic with recursive procedures and many other important topics have been discussed in detail in [1]. Our work begins with [16] and [1].

In modern days, programs are written in segments with procedures, which make the programs shorter in size and logically structured, and increases the reusability of code. So it is important to use procedures and heap operations (use of shared mutable resources) both in a single program. Verifying programs with such features is the main motivation of our work.

A logical system for software verification is called complete if every true judgment can be derived from that system. It ensures the strength of our system so that no further development is necessary for the logical system. If all true asserted programs are provable in Hoare's system where all true assertions are provided, we call it a relatively complete system. We will show the relative completeness of our system. A language is expressive if the weakest precondition can be defined in the language. We will also show that our language of specification is expressive for our programs. Relative completeness is discussed vastly in [1,6]. In this paper, relative completeness is sometimes paraphrased as completeness when it is not ambiguous.

The main contributions of our paper are as follows:

(1) A new logical system for verification of pointer programs and recursive procedures.
(2) Proving the soundness and the completeness theorems.
(3) Proving that our assertion language is expressive for our programs.

We know that Hoare's logic with recursive procedures is complete [1]. We also know that Hoare's logic with separation logic is complete [16]. But we do not know if Hoare's logic and separation logic for recursive procedures is complete.

In order to achieve our contributions, we will first construct our logical system by combining the axioms and inference rules of [1] and [16]. Then we will prove the expressiveness by coding the states in a similar way to [16]. At last we will follow a similar strategy in [1] to prove the completeness.

Although one may feel it easy to combine these two logical systems to achieve such a complete system, in reality it is not the case. Now we will discuss some challenges we face to prove its relative completeness.

(1) The axiom (AXIOM 9: INVARIANCE AXIOM), which is essential in [1] to show completeness, is not sound in separation logic.

(2) In the completeness proof of the extension of Hoare's logic for the recursive procedures in [1], the expression $\vec{x} = \vec{z}$ ($\vec{x}$ are all program variables and $\vec{z}$ are fresh) is used to describe the complete information of a given state in a precondition. A state in Hoare's logic is only a store, which is a function assigning to each variable a value. In separation logic, a state is a pair of a store and a heap. So the same expression cannot be used for a similar purpose for a heap, because a store information may contain variables $x_1, \ldots, x_m$ which are assigned $z_1, \ldots, z_m$ respectively, while a heap information consists of the set of the physical addresses only in the heap and their corresponding values. The vector notation cannot express the general information of the size of the heap and its changes because of allocation and deallocation of memory cells.

(3) Another challenge is to utilize the strongest postcondition of a precondition and a program. In case a program aborts in a state for which the precondition is valid, the strongest postcondition of the precondition and the program does not exist. But utilizing the strongest postcondition is necessary for completeness proof, because the completeness proof of [1] depends on it.

Now it is necessary to solve these obstacles for the proof of the completeness of our system. That is why it is quite challenging to solve the completeness theorem which is our principal goal.

The solutions to the challenges stated above are as follows:

(1) We will give an inference rule (inv–conj) as an alternative to the axiom (AXIOM 9: INVARIANCE AXIOM) in [1]. It will accept a pure assertion which does not have a variable common to the program. We will also give an inference rule (exists) that is analogous to the existential introduction rule in the first-order predicate calculus. We will show that the inference rule (RULE 10: SUBSTITUTION RULE I) in [1] is derivable in our system. Since the inference rules (RULE 11: SUBSTITUTION RULE II) and (RULE 12: CONJUNCTION RULE) in [1] are redundant in our system, we will remove them. As a result, the set of our axioms and inference rules will be quite different from the union of those of [1] and [16].

(2) We will give an appropriate assertion to describe the complete information of a given state in a precondition. Beside the expression $\vec{x} = \vec{z}$ for the store information, we will additionally use the expression $\mathrm{Heap}(x_h)$ for the heap information, where x_h keeps a natural number that is obtained by a coding of the current heap.

(3) For pointer programs, it is difficult to utilize the strongest postcondition because it is impossible to assert a postcondition for A and P where P may abort in a state for which A is true. We use $\{A\}P\{\mathrm{True}\}$ as the abort-free condition of A