Journal of Logical and Algebraic Methods in Programming 89 (2017) 41–66

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Revisiting concurrent separation logic [☆]

Pedro Soares [a,*], António Ravara [b,**], Simão Melo de Sousa [c,**]

[a] Universidade do Porto, Portugal
[b] CITI & DI-FCT, Universidade Nova de Lisboa, Portugal
[c] LISP & LIACC & DI-FE, Universidade da Beira Interior, Portugal

ARTICLE INFO

Article history:
Received 28 June 2016
Received in revised form 6 January 2017
Accepted 17 February 2017
Available online 27 February 2017

Keywords:
Concurrent separation logic
Structural operational semantics
Soundness proof

ABSTRACT

We present a new soundness proof of Concurrent Separation Logic (CSL) based on a structural operational semantics (SOS). We build on two previous proofs and develop new auxiliary notions to achieve the goal. One uses a denotational semantics (based on traces). The other is based on SOS, but was obtained only for a fragment of the logic — the Disjoint CSL — which disallows modifying shared variables between concurrent threads. In this work, we lift such a restriction, proving the soundness of full CSL with respect to a SOS. Thus contributing to the development of tools able of ensuring the correctness of realistic concurrent programs. Moreover, given that we used SOS, such tools can be well-integrated in programming environments and even incorporated in compilers.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

The aim of this work is to present a new soundness proof for Concurrent Separation Logic [1], with respect to a structural operational semantics [2]. This work adapts and extends the results presented by Brookes [3] and by Vafeiadis [4].

The axiomatic verification of programs goes back to Hoare Logic [5]. This seminal work introduces two key ideas, i) the specification of programs by means of what is known by a Hoare triple: $\{P\}C\{Q\}$, where P and Q are first order formulae, called the precondition and postcondition respectively, and C is an imperative program; ii) a deductive proof system to ensure the partial correctness of programs. A program is partially correct, if every execution of C from a state respecting the precondition does not abort and when it terminates the postcondition holds for its final state. The state for this logic is formed only by the store, i.e. a partial function that records the value of each variable. Hoare's work gave rise to numerous deductive systems, for instance the Owicki–Gries method [6,7] and Separation Logic [8,9].

The Owicki–Gries method is one of the first attempts to give a resource sensitive proof system for concurrent programs. To do this, Owicki and Gries augmented the programming language with i) parallel composition, $C \parallel C$; ii) local resources, resource r in C; and iii) a critical region, with r when B do C, where r denotes a resource. Each resource has a mutual exclusion lock, an assertion, called invariant, and a set of variables, called protected variables.

The execution of parallel composition non-deterministically chooses one of the commands to execute first. As usual, the parallel execution is assumed to be weakly fair, i.e. if a command is continually available to be executed, then this command

[☆] This work was partially funded by Fundação para a Ciência e a Tecnologia through AVIACC project, grant PTDC/EIA-CCO/117590, and CITI/FCT/UNL, grant Pest-OE/EEI/UI0527/2014.

* Principal corresponding author.
** Corresponding authors.
E-mail address: ptcsoares@fc.up.pt (P. Soares).

http://dx.doi.org/10.1016/j.jlamp.2017.02.004
2352-2208/© 2017 Elsevier Inc. All rights reserved.

will be eventually selected. The resource command declares a local variable r to be used in C. The critical region command waits for the availability of the resource r, and when B holds, it acquires r and starts the execution of C; the resource r is released upon the execution of C terminates.

The programs derivable by the Owicki–Gries method have to preserve the resource invariants when the resource is free, and respect the protection of variables by resources, i.e. a program needs to acquire all resources protecting a variable, before the program can change that variable. The parallel rule proposed by Owicki [6] requires that every variable occurring in the derivation proof of one command cannot be changed by another command, except for variables protected by a resource such that the variables only appear inside the critical region's proof. Thus, the Owicki–Gries method is not compositional.

Separation Logic (SL) supports reasoning about imperative programs with shared mutable data and consequently about dynamical data structures, such as lists and trees. In order to do this, the assertion and program languages used by Hoare had to be augmented. The assertions are extended with the constructs $\mathtt{emp}$, the empty memory; $e \mapsto e'$, a single memory cell e with the value e'; and $P * Q$, two disjoint memory's parts such that one satisfies P and the other satisfies Q. In this setting, the memory is usually represented by the heap — a partial function from the set of locations to the set of values. The store and the heap together define the state of a program.

The programing language is augmented with commands for memory manipulation. Naturally, the proof system is also extended with a rule for each new command and with a frame rule, used to enlarge the portion of memory considered in the condition of a specification. This rule is crucial to achieve local reasoning: program specifications only need to consider the relevant memory for their execution. Therefore, this local reasoning mechanism can be used to establish the partial correctness of disjoint concurrent programs, i.e. concurrent program which does not change shared variables.

In order to prove the soundness of the frame rule, and thus of local reasoning, it is sufficient to ensure the validity of two key properties: safety monotonicity and the frame property. Safety monotonicity states that if an execution does not abort for a given memory portion, then the execution does not abort for any memory portion that contains the initial one. The frame property says that if a command does not abort for a given memory portion, then every execution on a larger memory corresponds to an execution on the initial memory.

Recently provers based on separation logic were adopted in real industrial projects, Facebook's infer being the most prominent of such tools [10].

Since the introduction of SL, different authors adapted it to the verification of concurrent programs. Vafeiadis and Parkinson introduced RGSep, combining SL with Rely/Guarantee reasoning [11]. Reddy and Reynolds introduced a syntactic control of interference in SL [12], borrowing ideas from works on fractional permissions [13]. O'Hearn proposed Concurrent Separation Logic (CSL), combining SL with the Owicki–Gries method [1]. Brookes formalized CSL, extending the traditional Hoare triples with a resource context Γ and a rely-set A, what leads to specifications of the form $\Gamma \models_A \{P\}C\{Q\}$. A resource context records the invariant and the protected variables of each resource. A rely-set consists of all variables relevant for its derivation tree. This set ensures that CSL is a compositional proof method, proved sound with respect to a denotational semantics based on traces, where a program state is represented by a store, a heap and sets of resources, expressing resource ownership [3]. Actually, the rely-set was introduced after Wehrman and Berdine discovered a counter-example to the initial version of CSL [14], and it is analogous to the set of variables used by Owicki and Gries to check non-interference in their parallel rule.

Alternatively, Vafeiadis proposed a structural operational semantics (SOS) for concurrent programs synchronizing via resources, and proved the soundness of a part of CSL, the Disjoint CSL (DCSL) [4]. DCSL and CSL have different side conditions for the parallel rule. Concurrent threads, in DCSL, must not modify all variables that appear in other Hoare triples, however concurrent threads, in CSL, cannot modify variables that belong to other rely-sets.

Our aim is to remove the disjointness condition and obtain a soundness proof using a SOS for the full CSL (Section 6). The goal is relevant because CSL has been adopted as the basis for most modern program logics, and it is a step in the development of more expressive provers well integrated in software development environments and compilers. Not only does it allow proving correct concurrent programs manipulating shared resources, but also provides techniques to equip compilers with mechanisms of detecting data-races. Concretely, the contributions of this work are the following:

1. A novel notion of environment transition that simulates actions made by other threads. We define it taking into account the rely-set, available resources and their invariants (Section 5.1). This relation is crucial to study the soundness of the parallel rule[1];
2. The resource configuration that expresses ownership. It is defined by three sets: owned resources, locked resources, and available resources (Section 4.1). A program state is formed by a store, a heap and a resource configuration. Brookes also used sets of resources to represent resources ownership [3];
3. Illustrative examples that we prove correct in CSL, showing the proof system's expressiveness (Section 3).

This paper is an extended version of [15]. We present herein more examples and sketches of the proofs of the results reported in the short paper (which does not present proofs). Further examples and proofs in full detail can be found in a technical report [16].

[1] Vafeiadis used a completely different notion of environment transition (in RGSep [11]).

Download English Version:

https://daneshyari.com/en/article/4951421

Download Persian Version:

https://daneshyari.com/article/4951421

Daneshyari.com