Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 276 (2011) 5–28

www.elsevier.com/locate/entcs

A Revisionist History of Concurrent Separation Logic

Stephen Brookes

Department of Computer Science
Carnegie Mellon University
Pittsburgh, USA

Abstract

Concurrent Separation Logic is a resource-sensitive logic for fault-free partial correctness of concurrent programs with shared mutable state, combining separation logic with Owicki-Gries inference rules, in a manner proposed by Peter O'Hearn. The Owicki-Gries rules and O'Hearn's original logic lacked compositionality, being limited to programs with a rigid parallel structure, because of a crucial constraint that "no other process modifies" certain variables, imposed as a side condition in the inference rule for conditional critical regions. In prior work we proposed a more general formulation of a concurrent separation logic using resource contexts, and we offered a soundness proof based on a trace semantics. Recently Ian Wehrman and Josh Berdine discovered an example showing that this soundness proof relies on a hidden assumption, tantamount to "no concurrent modification", so that the proposed logic also suffices only for rigid programs. Here we show that, with a natural and simple adjustment we can avoid this problem. The key idea is to augment each assertion with a "rely set" of variables, assumed to be unmodified by other processes, and adjust the inference rules to validate and take advantage of these assumptions. This revised concurrent separation logic is compositional, allowing rigid and non-rigid programs, and the extra constraints imposed by rely set requirements ensure soundness. At the same time, we relax the Owicki-Gries constraints on the use of critical variables, allowing variables to be protected by multiple resources and building into the logic a simpler, yet more general, protection discipline. In the revised logic, a process wanting to write to a shared variable must acquire all resources that protect it, while a process wishing to read a shared variable need only acquire one such resource. This generalization brings concurrent separation logic closer in spirit to permission-based logics, in which processes may be allowed to perform concurrent reads.

Keywords: concurrency, shared memory, denotational semantics, resources, separation logic

1 Introduction

Concurrent Separation Logic (CSL) is a resource-sensitive logic for reasoning about fault-free partial correctness of shared-memory concurrent programs. CSL combines separation logic, originally introduced in [10] by John Reynolds for reasoning about sequential pointer programs, with Owicki-Gries rules for pointer-free shared-memory programs [7], in a manner proposed by Peter O'Hearn [6]. The Owicki-Gries and O'Hearn logics lack compositionality, being limited to programs with rigid parallel structure, because of a static constraint that "no other process modifies" certain variables, imposed as a side condition in the rule for conditional critical regions.

doi:10.1016/j.entcs.2011.09.013

In prior work we formulated a more general concurrent separation logic [3] using resource contexts in an attempt to avoid these limitations, and we gave a soundness proof, using on a trace-based denotational semantics. A major feature in this development was a semantic formalization of O'Hearn's notion of "ownership transfer" based on resource invariants, and O'Hearn's principle that processes "mind their own business" [6]. Recently Ian Wehrman and Josh Berdine found a counterexample [12] showing that this soundness proof makes a hidden assumption, tantamount to "no other process modifies", leading to the realization that the soundness analysis of [3] only suffices for rigid programs.

We show here that, with a systematic natural adjustment to the prior formulation, we can develop a fully compositional concurrent separation logic that avoids this problem. The key idea is to augment the assertions of CSL with a "rely set", representing a set of variables assumed to be left unmodified by the "environment". By making this set an integral part of assertions, we avoid the need for a non-compositional side condition; we are able to properly account for the assumptions and guarantees that a process makes about modifications to shared variables, in a purely syntax-directed manner.

At the same time, we relax the Owicki-Gries constraints on the use of critical variables, allowing variables to be protected by multiple resources and building into the logic a simpler, more general, protection discipline. This brings concurrent separation logic closer in spirit to permission-based logics, in which processes may be allowed to perform concurrent reads [1,2].

Again using action trace semantics, we sketch a soundness proof for the revised logic, this time without the hidden assumption and without requiring rigid program structure. We offer a series of examples, addressing Wehrman's problem, and showing that the augmented logic can deal with a wider variety of programs than the original, because of our relaxation of the Owicki-Gries constraints. We intend the revised and augmented logic presented here to replace the original.

We assume familiarity with separation logic, as defined by Reynolds [10].

2 Syntax

The syntax of our programming language (as in [3]) is given by the following abstract grammar, in which c ranges over the set **Com** of commands.

$$\begin{aligned} c ::= {} & \mathbf{skip} \mid i{:=}e \mid i{:=}[e] \mid [e]{:=}e' \mid i{:=}\mathbf{cons}\, E \mid \mathbf{dispose}\, e \\ \mid {} & c_1; c_2 \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{while}\ b\ \mathbf{do}\ c \\ \mid {} & \mathbf{with}\ r\ \mathbf{when}\ b\ \mathbf{do}\ c \mid c_1 \| c_2 \end{aligned}$$

Let e, b range over integer expressions and boolean expressions, respectively, and E range over list expressions of form $[e_1, \ldots, e_n]$. Expressions are *pure*, i.e. independent of the heap.

We distinguish syntactically between *identifiers* ($i \in \mathbf{Ide}$) denoting integer vari-