Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 336 (2018) 241–256

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Game Semantics of Concurrent Separation Logic

Paul-André Melliès[a] Léo Stefanesco[b]

[a] IRIF, CNRS, Université Paris Diderot

[b] École Normale Supérieure de Lyon

Abstract

In this paper, we develop a game-theoretic account of concurrent separation logic. To every execution trace of the Code confronted to the Environment, we associate a specification game where Eve plays for the Code, and Adam for the Environment. The purpose of Eve and Adam is to decompose every intermediate machine state of the execution trace into three pieces: one piece for the Code, one piece for the Environment, and one piece for the available shared resources. We establish the soundness of concurrent separation logic by interpreting every derivation tree of the logic as a winning strategy of this specification game.

Keywords: Concurrent separation logic, game semantics specification logic

1 Introduction

Concurrent separation logic (CSL) is an extension of Reynold's separation logic [12] formulated by O'Hearn [10] to establish the correctness of concurrent imperative programs with shared memory and locks. This specification logic enables one to establish the good behavior of these programs in an elegant and modular way, thanks to the frame rule of separation logic. A sequent of concurrent separation logic

$$r_1 : P_1, \ldots, r_n : P_n \vdash \{P\}\, C\, \{Q\}$$

consists of a Hoare triple $\{P\}C\{Q\}$ together with a context $\Gamma = r_1 : P_1, \ldots, r_n : P_n$ which declares a number of *resource variables* r_k (or mutexes) together with the CSL formula P_k which they satisfy as invariant. The validity of the program logic relies on a soundness theorem, which states that the existence of a derivation tree in concurrent separation logic

$$\dfrac{\begin{array}{c}\pi\\ \vdots\end{array}}{r_1 : P_1, \ldots, r_n : P_n \vdash \{P\}\, C\, \{Q\}}$$

ensures (1) that the concurrent program C will not produce any race condition at execution time, and (2) that the program C will transform every initial state

https://doi.org/10.1016/j.entcs.2018.03.026

1571-0661/© 2018 Published by Elsevier B.V.

satisfying P into a state satisfying Q when it terminates, *as long as* each resource r_k allocated in memory satisfies the CSL invariant P_k. The soundness of the logic was established by Brookes in his seminal papers on the trace semantics of concurrent separation logic [5,6]. His soundness proof was the object of great attention in the community, and it was revisited in a number of different ways, either semantic [13], syntactic [2] or axiomatic [7] and formalised in proof assistants. One main technical challenge in all these proofs of soundness is to establish the validity of the concurrent rule:

$$\frac{\Gamma \vdash \{P_1\}\, C_1\, \{Q_1\} \qquad \Gamma \vdash \{P_2\}\, C_2\, \{Q_2\}}{\Gamma \vdash \{P_1 * P_2\}\, C_1 \parallel C_2\, \{Q_1 * Q_2\}} \quad \text{Concurrent Rule}$$

and of the frame rule:

$$\frac{\Gamma \vdash \{P\}\, C\, \{Q\}}{\Gamma \vdash \{P * R\}\, C\, \{Q * R\}} \quad \text{Frame Rule}$$

In this paper, we establish the validity of these two rules (and of CSL at large) based on a new approach inspired by game semantics, which relies on the observation that the derivation tree π of CSL defines a winning strategy $[\pi]$ in a specification game. As we will see, the specification game itself is derived from the execution of the code C and its interaction with the environment (called the frame) using locks on the shared memory. The specification game expresses the usual rely-and-guarantee conditions as winning conditions in an interactive game played between Eve (for the code) and Adam (for the frame).

In the semantic proofs of soundness, two notions of "state" are usually considered, besides the basic notion *memory state* which describes the state of the variables and of the heap: (1) the *machine states* which are used to describe the execution of the code, and in particular include information about the status of the locks, and (2) the *logical states* which include permissions and other information invisible at the execution level, but necessary to specify the states in the logic. In particular, the tensor product $*$ of separation logic requires information on the permissions, and it is thus defined on logical states, not on machine states. The starting point of the paper is the observation that there exists a third notion of state, which we call *separated state*, implicitly at work in all the semantic proofs of soundness. A separated state describes which part of the global (logical) state of the machine is handled by each component interacting in the course of the execution. It is defined as a triple $(\sigma_C, \boldsymbol{\sigma}, \sigma_F)$ consisting of

- the logical state $\sigma_C \in \mathbf{LState}$ of the code,
- the logical state $\sigma_F \in \mathbf{LState}$ of the frame,
- a function $\boldsymbol{\sigma} : \{r_1, \ldots, r_n\} \to \mathbf{LState} + \{C, F\}$ which tells for every resource variable r whether it is locked and owned by the code, $\boldsymbol{\sigma}(r) = C$, locked and owned by the frame, $\boldsymbol{\sigma}(r) = F$, or available with logical state $\boldsymbol{\sigma}(r) \in \mathbf{LState}$.

This leads us to a "span"

$$\text{machine states} \xLeftarrow{\textit{refines}} \text{separated states} \xRightarrow{\textit{refines}} \text{logical states} \tag{1}$$