Refinement-Based Game Semantics with Concurrency
A Fork of `rbgs`

R. Philip Stetson IV

December 9, 2025

Abstract

The goal of this fork is to extend refinement-based game semantics (RBGS) with cooperative concurrency in a way that remains compositional and proof-oriented. Building on the three-dimensional refinement algebra of Zhang et al. [3], the cooperative threading discipline of Zhang et al. [2], and the compositional linearizability perspective of Vale et al. [1], we add: (i) a concurrency effect signature, free monad, and Rocq-friendly notation; (ii) a handler- and scheduler-parametric operational semantics over configurations; and (iii) a reachability-based refinement predicate and small driver lemmas. The intent is to expose precise interfaces (signatures, handlers, schedulers) that can be tightened into full refinement proofs and case studies.

1 Problem

RBGS models programs as strategies over effect signatures with refinement conventions that compose along modules, abstraction levels, and state components [3]. The missing piece is a *cooperative concurrency* layer—spawn, yield, send/recv, join—that:

- respects the categorical structure already present (polynomial signatures, free monads, symmetric monoidal products);
- exposes scheduler and memory relations (e.g., TKMR [2]) as parameters rather than hard-wiring an operational model; and
- supports compositional refinement in the spirit of three-dimensional algebra while remaining compatible with compositional linearizability [1].

2 Approach

Effect signature and monad. We introduce a concurrency effect signature

$$\mathsf{Conc}(X) ::= \mathsf{spawn} : X_{\mathsf{tid}} \mid \mathsf{yield} : 1 \mid \mathsf{send} : \mathsf{Chan} \times \mathsf{Val} \to 1 \mid \mathsf{recv} : \mathsf{Chan} \to \mathsf{Val} \mid \mathsf{join} : \mathsf{Tid} \to \mathsf{Val},$$

implemented as a dependent type and converted to a polynomial signature Σ_{Conc}. The free monad T_{Conc} (module `ConcMonad`) provides η, μ, and a tensor $m_{X,Y} : T_{\mathsf{Conc}}X \times T_{\mathsf{Conc}}Y \to T_{\mathsf{Conc}}(X \times Y)$ that currently sequences then pairs results (a conservative, compositional default awaiting richer interleavings). Smart constructors (`spawn`, `yield`, `send`, `recv`, `join`) and a light `do`-notation keep scripts concise.

Configurations, handlers, and scheduler. A configuration (`ConcStrategy`) is a list of thread states plus shared state; shared-state merge is abstracted as a binary operator. Handlers interpret spawn/yield/send/recv/join against shared state and optionally add runnable threads. A scheduler $\sigma : C \to \mathsf{option}\,\mathbb{N}$ selects a runnable thread. The small-step relation

$$c \xrightarrow{\sigma,h,t} c'$$

is defined by interpreting the selected thread through T_{Conc}, applying handlers, updating the chosen thread, and appending any spawned threads.

Refinement. The refinement predicate (`ConcRefinement`) relates an abstract term $t : T_{\mathsf{Conc}}\,C$ to a concrete configuration c by reachability:

$$R_{\mathsf{conc}}(t, c) \;:\Longleftrightarrow\; \exists c_0.\, t = \eta\, c_0 \;\wedge\; (c_0 \xrightarrow{\sigma,h,t} {}^{*} c).$$

This exposes scheduler and handler choices to the proof obligations and aligns with the three-dimensional algebra view: horizontal composition is list concatenation of threads, vertical composition is monadic substitution, and spatial composition is mediated by the shared-state merge.

Examples. Two driver files exercise the machinery: `ConcSpawn.v` shows that one scheduler step adds a spawned thread; `ConcChan.v` steps through a send/recv pair with unchanged shared state. They serve as sanity checks rather than full case studies.

3 Relationship to prior work

- **Three-dimensional refinement** ([3]): The reachability-based R_{conc} mirrors the algebraic composition across modules, abstraction levels, and state components, leaving scheduler/memory relations parametric.
- **TKMR and cooperative stacks** ([2]): Handlers are the hook for threaded Kripke memory relations; the next step is to instantiate them with TKMR-like relations and prove preservation under compilation passes.
- **Compositional linearizability** ([1]): The tensor and configuration merge provide the symmetric monoidal structure needed to connect to Karoubi-envelope reasoning; coherence laws remain to be proved for true parallel composition.

4 Status

- Implemented: `ConcSignature`, `ConcMonad` (with sequential tensor and smart constructors/do-notation), `ConcStrategy` (handlers/scheduler/step), `ConcRefinement` (reachability-based), examples with concrete handlers and scheduler steps.
- Limitations: tensor is sequential; no fairness or scheduler obligations are encoded; handlers are uninterpreted; refinement is reachability-only (no traces or liveness); examples are single-step sanity checks.
- Build: Coq 8.15.2, OCaml 4.12.1, `./configure -nocompcert && make -j4` (CompCert optional on Linux).

5 Next work

- Strengthen R_{conc} to trace/step-indexed simulations with fairness assumptions on σ; relate to TKMR-style memory relations.
- Replace the sequential tensor with a parallel tensor satisfying symmetric monoidal coherence and compatible with Karoubi envelopes; prove associativity/symmetry/unit laws at the configuration level.
- Instantiate handlers with TKMR to connect to CompCertOC-style compilation passes and prove preservation for spawn/yield/send/recv/join.
- Develop real case studies (e.g., producer/consumer, fork/join pipelines) with compositional refinement theorems and extracted traces.

References

[1] A. O. Vale, Z. Shao, and Y. Chen. A compositional theory of linearizability. *J. ACM*, 71(2), Apr. 2024.

[2] L. Zhang, Y. Wang, Y. Liang, and Z. Shao. Compcertoc: Verified compositional compilation of multi-threaded programs with shared stacks. *Proc. ACM Program. Lang.*, 9(PLDI), June 2025.

[3] Y. Zhang, J. Koenig, Z. Shao, and Y. Wang. Unifying compositional verification and certified compilation with a three-dimensional refinement algebra. *Proc. ACM Program. Lang.*, 9(POPL), Jan. 2025.