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

R. Philip Stetson IV

9 December 2025

Abstract

This note summarizes the mathematical setting of the refinement-based game semantics (RBGS) project and the contributions of this fork: a concurrency effect signature, its free monad, a configuration/strategy skeleton, and example drivers. The aim is to make concurrency compositional, refinement-friendly, and compatible with the categorical structure already present in RBGS.

1 Context and problem

RBGS models programs as strategies over effect signatures, equipped with refinement conventions that compose horizontally (modules), vertically (abstractions), and spatially (state components) [3]. The open problem addressed here is: *how do we add cooperative concurrency (spawn, yield, channel communication, join) while preserving the algebraic and categorical structure that underpins RBGS and downstream compilation correctness?*

2 Questions and commands to realize

The concurrency extension is framed by the following questions:

(Q1) How should a concurrency effect be presented so its algebra (sequencing, parallel composition) integrates with existing signatures?

(Q2) Which categorical structure captures concurrent composition so that refinement remains compositional?

(Q3) How can we stage scheduler choices and memory relations so the semantics is parametric yet implementable?

In operational terms, the "commands" we seek to realize are the effect operations:

$$\mathsf{spawn} : 1 \to \mathsf{Tid}, \quad \mathsf{yield} : 1 \to 1, \quad \mathsf{send} : \mathsf{Chan} \times \mathsf{Val} \to 1, \quad \mathsf{recv} : \mathsf{Chan} \to \mathsf{Val}, \quad \mathsf{join} : \mathsf{Tid} \to \mathsf{Val}.$$

3 Categorical and semantic foundations

Effect signatures. RBGS treats an effect signature as a polynomial endofunctor on **Set**. We add a concurrency signature Conc with the operations above; its presentation as a dependent type $\mathsf{Conc} : \mathsf{Type} \to \mathsf{Type}$ is converted to a polynomial Σ_{Conc} to align with existing machinery.

Free monad and tensor. The module `models/ConcMonad.v` builds the free monad T_{Conc} over Σ_{Conc}, inheriting unit η and multiplication μ from the general free-monad construction. We equip T_{Conc} with a tensor

$$m_{X,Y} : T_{\mathsf{Conc}}X \times T_{\mathsf{Conc}}Y \to T_{\mathsf{Conc}}(X \times Y),$$

realizing a simple parallel pairing. This aligns with the symmetric monoidal structure used throughout RBGS and sets the stage for more refined interleavings.

Strategies and configurations. The module `models/ConcStrategy.v` models a configuration as a list of thread states paired with shared state. System strategies $C \to T_{\mathsf{Conc}}C$ combine via the tensor and a configuration merge, anticipating a scheduler that selects interleavings. This mirrors the game-semantic view of plays as interactions over combined arenas [1].

Refinement scaffold. The module `models/ConcRefinement.v` states refinement obligations between abstract concurrent terms and concrete configurations. The current proofs are placeholders, but the structure follows the refinement-convention pattern of [3, 2]: simulations parameterized by scheduler and shared-memory relations (e.g., TKMR).

4 Implementation summary

- **Signature and monad:** `ConcSignature.v`, `ConcMonad.v`.
- **Strategies and refinement:** `ConcStrategy.v`, `ConcRefinement.v`.
- **Examples:** `examples/ConcSpawn.v`, `examples/ConcChan.v` as driver proofs.
- **Build:** Coq 8.15.2, OCaml 4.12.1, `./configure -nocompcert && make -j4` (CompCert optional on Linux).

5 Why this design

- *Compositionality*: By staying within polynomial signatures and free monads, concurrency composes with existing effects and categorical structure without bespoke encodings.
- *Refinement readiness*: The refinement scaffold mirrors the 3D algebra of [3], enabling horizontal (threads), vertical (abstraction), and spatial (state) composition once proofs are filled in.
- *Scheduler parametricity*: The strategy/configuration split leaves scheduling abstract, allowing instantiation with cooperative schedulers or refinement conventions as in [2].
- *Game-semantic alignment*: Parallel composition is expressed via monoidal structure on strategies, echoing the Karoubi-envelope discipline for avoiding neutral-element interference [1].

6 Next steps

- Strengthen R_{conc} to a step-indexed or trace-based simulation relating abstract terms to scheduled configurations; validate monad laws plus scheduler fairness.

- Instantiate scheduler and memory refinement conventions (e.g., TKMR-style relations) to obtain end-to-end theorems for spawn/yield/send/recv/join.
- Elevate the example stubs into full refinement case studies with compositional linking proofs and extracted traces.
- Tie the concurrency tensor to the existing symmetric monoidal structure to document coherence (associativity/symmetry/unit) at the strategy level.

References

[1] V. Authors. Compositional linearizability via karoubi envelopes. *Journal of the ACM*, 2024.

[2] Z. Shao and et al. Compcertoc: Cooperative multithreading with shared stacks. In *Proceedings of the 46th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2025.

[3] Z. Shao and et al. Compcertoe: A 3d refinement algebra for open-environment compilation. In *Proceedings of the 52nd ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2025.