# Automated Verification of Effectful OCaml code via Operational Game Semantics

Guilhem Jaber
Gallinette - LS2N - Université de Nantes
guilhem.jaber@inria.fr - http://guilhem.jaber.fr

Keywords: Functional Programming, Automated Reasoning, Static Analysis, Semantics of Programming Languages

## 1 Context

Automated verification of respectively higher-order functional programs and first-order imperative programs have been developed, based on techniques like (higher-order) model-checking and abstraction interpretation. In this setting, *compositional* verification has gained a large interest recently, in order to design techniques that scale to large codebases, as shown by the success of the Infer tool [1]. Indeed, compositionality allows modular reasoning on source code, analysing each function or each module independently, then combining the results.

But compositional verification of effectful higher-order programs, that can perform side effects using mutable memory or exceptions, is a challenging problem.

Operational game semantics provides a general framework to study such effectful higher-order languages [Lai07]. To do so, it represents programs as Labelled Transition Systems (*LTS*) that generate traces that correspond to interaction with any possible environment. Doing so, it provides a compositional representation of programs, by specifying the possible behaviors of the interacting environments.

Automating the reasoning on such LTS for an effectul fragment of OCaml has been recently explored, in the setting of contextual equivalence of programs [Jab20]. We seek here to generalize this work to go beyond proving equivalences, reasoning more generally on safety properties.

## 2 Subject

The goal of this internship is to design an automated technique to check safety of programs written in a effectful, higher-order fragment of OCaml. By safety, we mean checking if invariants annotated in the source-code via `assert` expressions indeed hold.

For sake of simplicity, this analysis will not take typing information into account. So rather than working directly with OCaml source code, it will target a fragment of Lambda, the intermediate language used by the OCaml compiler. It is an untyped $\lambda$-calculus with mutable memory blocks [2].

The internship will be organized into the following tasks:

- Characterize a subset of Lambda rich enough to express recursive programs that handle list, trees, global references and exceptions. We will take inspiration from Malfunction [Dol16].
- Design a symbolic execution of this fragment of Lambda.

[1] `https://fbinfer.com/`

[2] For more on it, see Chapter 25 of Real World OCaml (`https://dev.realworldocaml.org/compiler-backend.html`) and Chambart's talk at OCaml 2016 workshop (`https://www.youtube.com/watch?v=R3Uk9gt90Tk`)

- Using it, provide a symbolic representation of the LTS used to represent programs in operational game semantics. Here, we follow the general approach provided in [Jab20], that we will extend to deal with exceptions.

And if time permits:

- Encode reachability problems over this symbolic representation using higher-order constrained Horn clauses [CBOR17]
- Explore satisfiability of these clauses using existing solvers like the Spacer engine of z3 [3], Eldarica [4] or Hoice [5].

# 3 Expected Skills

We are looking for candidates with good skills in functional programming (ideally OCaml). Knowledge of either semantics of programming languages, automated verification or compilation will be appreciated.

# 4 Practical Informations

This internship is part of the CAVOC project funded by the Inria/Nomadic Labs partnership. The intern will receive a stipend ("gratification de stage") following the legal rate.

It may lead to a PhD thesis on the general topic of automated verification of OCaml programs, targeting especially its type system (GADTs, modules).

It will be located in the Inria Gallinette team of LS2N, on the Faculté des Sciences campus of the university of Nantes.

# References


[CBOR17] Toby Cathcart Burn, C.-H. Luke Ong, and Steven J. Ramsay. Higher-order constrained Horn clauses for verification. *Proc. ACM Program. Lang.*, 2(POPL), December 2017.

[Dol16] Stephen Dolan. Malfunctional programming. In *ML Workshop*, 2016.

[Jab20] Guilhem Jaber. SyTeCi: Automating contextual equivalence for higher-order programs with references. *Proc. ACM Program. Lang.*, 4(POPL), December 2020.

[Lai07] Jim Laird. A fully abstract trace semantics for general. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, page 667–679, Berlin, Heidelberg, 2007. Springer-Verlag.


[3] https://github.com/Z3Prover/z3/

[4] `https://github.com/uuverifiers/eldarica`

[5] `https://github.com/hopv/hoice`