Compositional Automated Verification of OCaml

Guilhem Jaber
Gallinette Team, LS2N, Univ. Nantes

Journée scientifique Inria - Nomadic Labs
21st of Septembre

Goal

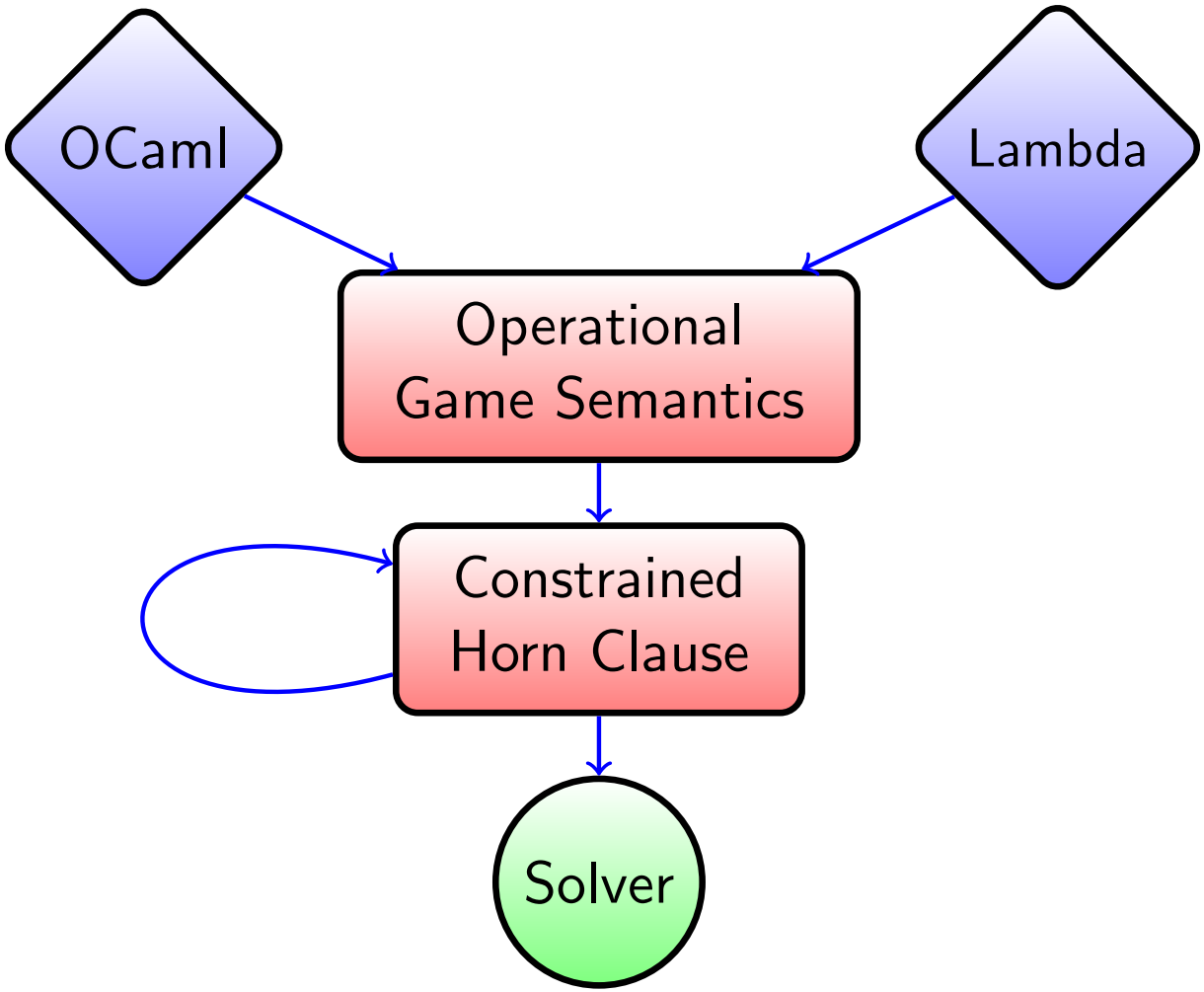
- Checking automatically that
  - invariants specified via assert expressions in the code holds;
  - there are no uncaught exceptions
- compositionally
  - by reasoning on any code that uses the analyzed module
- taking into account the abstractions enforced by the language:
  - local references
  - parametricity
  - abstract data-types
  - invariants via GADTs

```
module M : sig
  type t
  val get : unit → t
  val check : t → unit
end = struct
  type t = int
  let c = ref 0
  let get () = c :=!c + 1; !c
  let check x = assert (x > 0)
end
```

Is there some OCaml code that uses this module and invalidate the assert ?

Analyzed languages

- Fragment of OCaml with:
  - Exceptions
  - Mutable memory (references, mutable records)
  - Polymorphism (Hindley-Milner with value restriction, Higher-rank via records)
  - Functors (both applicative and generative)
  - GADTs
  - Concurrency (Lwt)
- Fragment of Lambda
  - untyped intermediate language used by the compiler
  - restriction à la Malfunction


OCaml
Lambda
Operational
Game Semantics
Constrained
Horn Clause
Solver

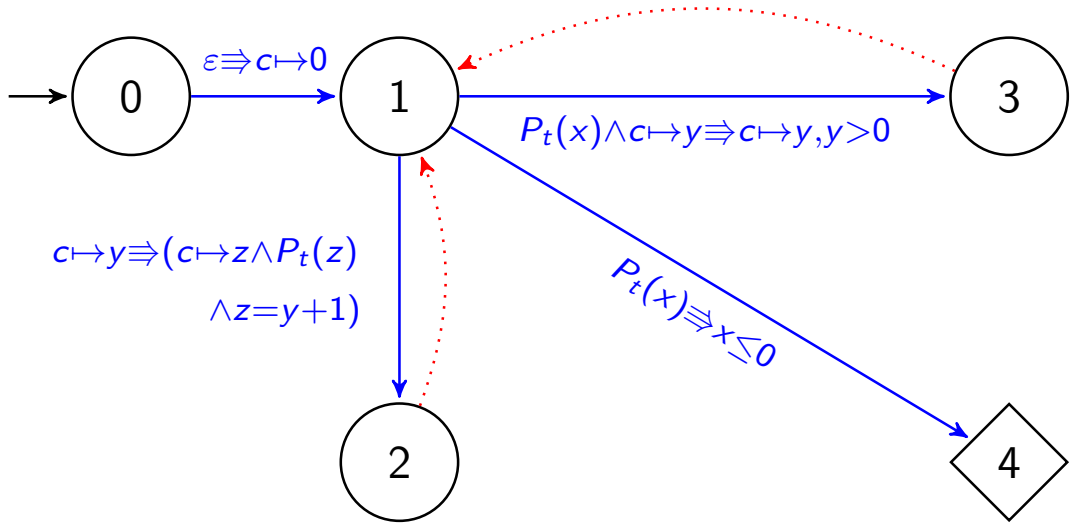
Operational Game Semantics

- An open program by is represented by:

  a labelled transition system whose traces are interactions between the program and any environment.

- values exchanged between the program and the environment are abstracted
  - $\rightsquigarrow$ enforces parametricity
- symbolic representation of the LTS
- "Failed states" to represent unsafe interactions

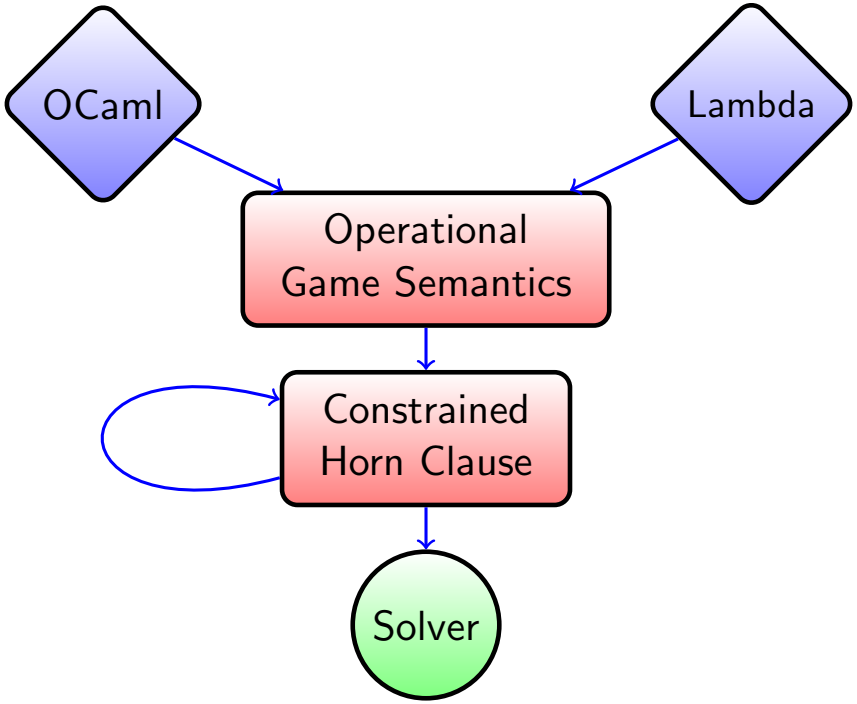
```
module M : sig
  type t
  val get : unit → t
  val check : t → unit
end = struct
  type t = int
  let c = ref 0
  let get () = c := !c + 1; !c
  let check x = assert (x > 0)
end
```



Constrained Horn Clauses

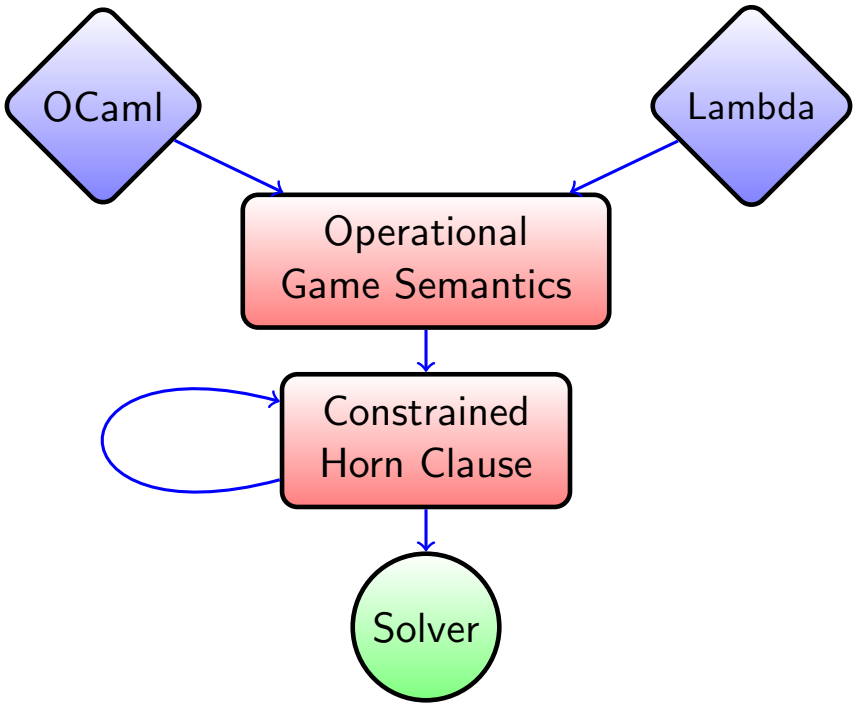
- First-order formula
  - with at most one positive occurrence of an uninterpreted predicate
  - written in a combination of theories (linear arithmetic, arrays, ...)
- Solvers for checking satisfiability: z3, Eldarica;
- Used for automated verification of various programming languages
  - C (SeaHorn), Java (JayHorn), Ada (AdaHorn)
- Used for checking contextual equivalence for a fragment of OCaml (SyTeCi)
- Higher-order CHC
  - Needed for higher-order recursion;
  - Reduction to first-order CHC via defunctionalization (Ong et. al)
- Simplification of the theories used
  - Abstract interpretation based transformations of CHC (Monniaux & Gonnord)

Organization



- PhD student to work on Operational Game Semantics for the OCaml type system (GADTs, Module)
- 2years postdoc to work on the Constrained Horn Clause part.

Organization



- PhD student to work on Operational Game Semantics for the OCaml type system (GADTs, Module)
- 2years postdoc to work on the Constrained Horn Clause part.

Let's talk about static analysis of OCaml on the Slack channel
#ocaml-static-analysis on tezos-dev