Salto: Static Analyses for Trustworthy OCaml

Benoît Montagu (lead), Thomas Genet, Thomas Jensen

Inria

Inria+Nomadic Labs Scientific Day — June 1st 2022 — Inria Paris

Salto project

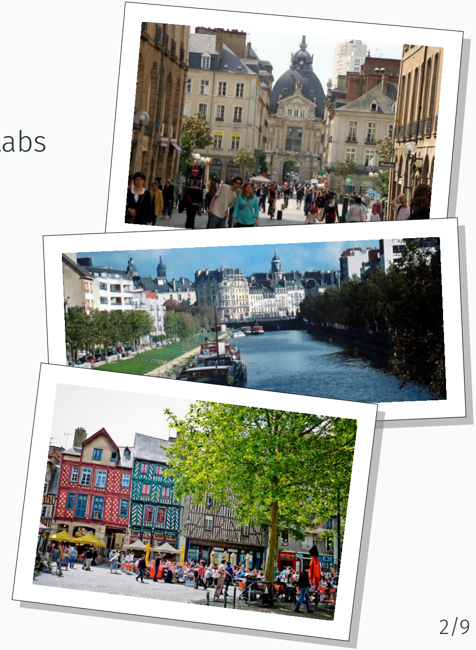
What: static analysis for OCaml programs

Where: Celtique research team, Inria Rennes

Who: B. Montagu + T. Genet + T. Jensen + Nomadic Labs

Accepted: in summer 2021

Starting: as soon as possible!



We are *hiring* a research engineer
on a 2 year contract

Contact me!

✉ benoit.montagu@inria.fr

Static Analysis of OCaml Programs

Goals

- Detect the most common mistakes in OCaml programs, that cannot already be caught by the type system
- Improve the safety of Nomadic Labs' code by detecting actual issues
- Make the analyser available to the OCaml ecosystem

Goals

- Detect the most common mistakes in OCaml programs, that cannot already be caught by the type system
- Improve the safety of Nomadic Labs' code by detecting actual issues
- Make the analyser available to the OCaml ecosystem

Examples

- Uncaught exceptions (e.g., system calls)

```
Fatal error: exception Sys_error("foo.txt: No such file or directory")
```

Static Analysis of OCaml Programs

Goals

- Detect the most common mistakes in OCaml programs, that cannot already be caught by the type system
- Improve the safety of Nomadic Labs' code by detecting actual issues
- Make the analyser available to the OCaml ecosystem

Examples

- Uncaught exceptions (e.g., system calls)

  `Fatal error: exception Sys_error("foo.txt: No such file or directory")`

- Violated preconditions (e.g., non-empty list)

  `Fatal error: exception Failure("hd")`

Static Analysis of OCaml Programs

Goals

- Detect the most common mistakes in OCaml programs, that cannot already be caught by the type system
- Improve the safety of Nomadic Labs' code by detecting actual issues
- Make the analyser available to the OCaml ecosystem

Examples

- Uncaught exceptions (e.g., system calls)

  `Fatal error: exception Sys_error("foo.txt: No such file or directory")`
- Violated preconditions (e.g., non-empty list)

  `Fatal error: exception Failure("hd")`
- Failure of an assertion that was explicitly written by a programmer

  `Fatal error: exception Assert_failure("bar.ml", 6, 14)`

Goals

- Detect the most common mistakes in OCaml programs, that cannot already be caught by the type system
- Improve the safety of Nomadic Labs' code by detecting actual issues
- Make the analyser available to the OCaml ecosystem

Examples

- Uncaught exceptions (e.g., system calls)

  `Fatal error: exception Sys_error("foo.txt: No such file or directory")`
- Violated preconditions (e.g., non-empty list)

  `Fatal error: exception Failure("hd")`
- Failure of an assertion that was explicitly written by a programmer

  `Fatal error: exception Assert_failure("bar.ml", 6, 14)`
- Arithmetic errors (e.g., off-by-one, overflows)

  `Fatal error: exception Invalid_argument("index out of bounds")`

What We Achieved So Far

- An analyser for a subset of OCaml (pure, exception-less, shallow patterns…)
  - This is already an interesting challenge! (untyped, higher-order language)

What We Achieved So Far

- An analyser for a subset of OCaml (pure, exception-less, shallow patterns...)
  - ☞ This is already an interesting challenge! (untyped, higher-order language)
- **An abstract interpretation-based control-flow analysis (∇CFA)**

Benoît Montagu and Thomas P. Jensen. "Trace-based control-flow analysis". In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*. Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 482–496. DOI: 10.1145/3453483.3454057

What We Achieved So Far

- An analyser for a subset of OCaml (pure, exception-less, shallow patterns...)
  - ☞ This is already an interesting challenge! (untyped, higher-order language)
- **An abstract interpretation-based control-flow analysis (∇CFA)**

Benoît Montagu and Thomas P. Jensen. “Trace-based control-flow analysis”. In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*. Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 482–496. DOI: 10.1145/3453483.3454057

- ∇CFA uses a local fixpoint solver [1, 7, 9] performing *widened* Kleene iterations

```
val fix: ((X.t -> Y.t) -> (X.t -> Y.t)) -> (X.t -> Y.t)
```

  - ☞ Cleanly separates the declaration of transfer functions from the iteration/convergence strategy
  - ☞ Advocated by B. Jeannet [3], and actually used in the `goblint` C analyser [10]

What We Achieved So Far

- An analyser for a subset of OCaml (pure, exception-less, shallow patterns...)
  - ☞ This is already an interesting challenge! (untyped, higher-order language)
- **An abstract interpretation-based control-flow analysis (∇CFA)**

Benoît Montagu and Thomas P. Jensen. "Trace-based control-flow analysis". In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211.* Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 482–496. DOI: `10.1145/3453483.3454057`

- ∇CFA uses a local fixpoint solver [1, 7, 9] performing *widened* Kleene iterations

```
val fix: ((X.t -> Y.t) -> (X.t -> Y.t)) -> (X.t -> Y.t)
```

  - ☞ Cleanly separates the declaration of transfer functions from the iteration/convergence strategy
  - ☞ Advocated by B. Jeannet [3], and actually used in the `goblint` C analyser [10]
- ∇CFA exploits an abstract domain for regular sets of algebraic values:
  - ☞ Draws some ideas from tree automata
  - ☞ Reuses standard arithmetic abstract domains
  - ☞ Inspired from the *type graphs* (cyclic structures, used in analysis of Prolog [2])
  - ☞ Inspired from the theory of equi-recursive types

An OCaml Example: Insertion In a Sorted List

```
(* absolute value *)
let abs x = if x >= 0 then x else -x

(* insertion in a sorted list *)
let rec insert x l = match l with
| []  ->      [x]
|                           y :: l'  ->
  if x < y then      x :: l  else      y :: insert x l'


(* arbitrary sorted list of size n whose elements are >= 42 *)
let rec make n =
  if n <= 0 then []  else insert (42 + (Random.int max_int)) (make (n-1))

let head = match insert 1 (make (Random.int max_int)) with
| []  -> -127 (* /!\ should never happen *)
|                           x :: _  -> x     (* head of the list: must be 1 *)

let result = (head = 1) (* should always return true *)
```

The Same Example, Translated

```
(* absolute value *)
val abs x = if x >= 0 then x else -x

(* insertion in a sorted list *)
val rec insert x l = match l with
| Nil -> Cons (x, Nil)
| Cons p -> match p with (y,   l') ->
  if x < y then Cons (x,   l) else Cons (y,   insert x l')
end end

(* arbitrary sorted list of size n whose elements are >= 42 *)
val rec make n =
  if n <= 0 then Nil else insert (42 + (abs           ?int)) (make (n-1))

val head = match insert 1 (make (abs           ?int)) with
| Nil -> -127 (* /!\ should never happen *)
| Cons p -> match p with (x,   _) -> x end (* head of the list: must be 1 *)
end
val result = (head = 1) (* should always return true *)
```

The Same Example, Analysed

Example programs: insert_sorted_list

```
(* absolute value *)
val abs x = if x >= 0 then x else -x

(* insertion in a sorted list *)
val rec insert x l = match l with
| Nil -> Cons (x, Nil)
| Cons p -> match p with (y,   l') ->
  if x < y then Cons (x,   l) else Cons (y,   insert x l')
end end

(* arbitrary sorted list of size n whose elements are >= 42 *)
val rec make n =
  if n <= 0 then Nil else insert (42 + (abs           ?int)) (make (n-1))

val head = match insert 1 (make (abs           ?int)) with
| Nil -> -127 (* /!\ should never happen *)
| Cons p -> match p with (x,   _) -> x end (* head of the list: must be 1 *)
end
val result = (head = 1) (* should always return true *)
```

Analyse!

Output:

```
{ bools = { true } }

(* Running time: 654ms *)
(* Statistics:
   states: 76; edges: 131; max fanout (median): 20 (0); iterations: 5 *)
```

- The analyser runs natively, and in the web browser thanks to `js_of_ocaml`
- On this program, the analyser infers the most precise result
- And emits no warning

The Same Example, Analysed

Example programs: insert_sorted_list

```
1  (* absolute value *)
2  val abs x = if x >= 0 then x else -x
3
4  (* insertion in a sorted list *)
5  val rec insert x l = match l with
6  | Nil -> Cons (x, Nil)
7  | Cons p -> match p with (y,   l') ->
8    if x < y then Cons (x,   l) else Cons (y,   insert x l')
9  end end
10
11 (* arbitrary sorted list of size n whose elements are >= 42 *)
12 val rec make n =
13   if n <= 0 then Nil else insert (42 + (abs           ?int)) (make (n-1))
14
15 val head = match insert 1 (make (abs           ?int)) with
16 | Nil -> -127 (* /!\ should never happen *)
17 | Cons p -> match p with (x,   _) -> x end (* head of the list: must be 1 *)
18 end
19 val result = (head = 1) (* should always return true *)
20
```

Analyse!

Output:

```
{ bools = { true } }

(* Running time: 654ms *)
(* Statistics:
   states: 76; edges: 131; max fanout (median): 20 (0); iterations: 5 *)
```

- The analyser runs natively, and in the web browser thanks to `js_of_ocaml`
- On this program, the analyser infers the most precise result
- And emits no warning

Some remarks:

- Line 19: `result` is necessarily the boolean `true`
- Line 16 is detected as unreachable
  ☞ it could be deleted!
- Line 15: the list `insert 1 (make (abs ?int))` necessarily starts with the value 1

The Same Example, Analysed

Example programs: insert_sorted_list

```
1  (* absolute value *)
2  val abs x = if x >= 0 then x else -x
3
4  (* insertion in a sorted list *)
5  val rec insert x l = match l with
6  | Nil -> Cons (x, Nil)
7  | Cons p -> match p with (y,   l') ->
8    if x < y then Cons (x,   l) else Cons (y,   insert x l')
9  end end
10
11 (* arbitrary sorted list of size n whose elements are >= 42 *)
12 val rec make n =
13   if n <= 0 then Nil else insert (42 + (abs           ?int)) (make (n-1))
14
15 val head = match insert 1 (make (abs           ?int)) with
16 | Nil -> -127 (* /!\ should never happen *)
17 | Cons p -> match p with (x,   _) -> x end (* head of the list: must be 1 *)
18 end
19 val result = (head = 1) (* should always return true *)
20
```

Analyse!

Output:

```
{ bools = { true } }

(* Running time: 654ms *)
(* Statistics:
   states: 76; edges: 131; max fanout (median): 20 (0); iterations: 5 *)
```

- The analyser runs natively, and in the web browser thanks to `js_of_ocaml`
- On this program, the analyser infers the most precise result
- And emits no warning

Some remarks:

- Line 19: `result` is necessarily the boolean `true`
- Line 16 is detected as unreachable
  it could be deleted!
- Line 15: the list `insert 1 (make (abs ?int))` necessarily starts with the value 1

Tested on plenty of examples (higher-order, monadic, CPS, ill-typed...)

The Road Ahead (1)

Support more features found in OCaml

- Support exceptions
  - Exceptions as values
  - Local exceptions (difficult inside recursive functions)
- Detect arithmetic overflows/underflows for `Int31`, `Int32`, `Int63`, `Int64`
- Support mutable state
  - References and mutable data-types
  - Arrays
  - External state provided by the OS (e.g., file descriptors)
- Cyclic values, e.g.: `let rec l = 1 :: l`
- Labelled arguments, modules, functors, objects, classes...

The Road Ahead (2)

Refine the analysis

- Extend the fixpoint solver to interleave forward and backward analyses
  - ☞ Asks to find a pair of (post-)fixpoints for two mutually-defined functionals
- Incorporate a narrowing phase to the fixpoint solver
- Exploit the types inferred by the OCaml compiler (reduced product)
- Specific abstract domains for strings, bytes, sets, maps, hash-tables...

The Road Ahead (3)

Improve efficiency

- Transition from a simple/naive implementation to an efficient/clever one

Improve usability

- Handle the actual OCaml AST as input
  - A rather large AST...
  - ... that contains some redundant features
  - ... and may change over time
- Discover the structure of an OCaml project
  - Start some work on improving `dune describe`
- Make the analyser incremental
  - Minimise the number of necessary re-computations
  - Save partial results on the filesystem

This is where a research engineer would be extremely useful!

The Road Ahead (4)

Long term challenges

- Relational analysis (especially: input/output relations)
  - Stable relations

  Benoît Montagu and Thomas P. Jensen. “Stable Relations and Abstract Interpretation of Higher-order Programs”. In: *Proc. ACM Program. Lang.* 4.ICFP (2020), 119:1–119:30. DOI: 10.1145/3409001

  - S. Bautista and T. Losekoot Ph.D theses on relational domains
- Non-sequential code (LWT/Async)

The Road Ahead (4)

Long term challenges

- Relational analysis (especially: input/output relations)
  - ☞ Stable relations
  - Benoît Montagu and Thomas P. Jensen. "Stable Relations and Abstract Interpretation of Higher-order Programs". In: *Proc. ACM Program. Lang.* 4.ICFP (2020), 119:1–119:30. DOI: `10.1145/3409001`
  - ☞ S. Bautista and T. Losekoot Ph.D theses on relational domains
- Non-sequential code (LWT/Async)

Much longer term challenges

- Low-level representation of data (`Obj` module)
- Polymorphic and physical equality
- Signals? Algebraic effects (one-shot continuations)? Multicore?

☞ For "*Salto 2*"?

Thank you for your attention

Salto: Static Analyses for Trustworthy OCaml

B. Montagu + T. Genet + T. Jensen + Nomadic Labs

Celtique research team, Inria Rennes

- Baudouin L. Charlier and Pascal Van Hentenryck. *A Universal Top-Down Fixpoint Algorithm*. Tech. rep. USA, 1992. URL: ftp://ftp.cs.brown.edu/pub/techreports/92/cs92-25.pdf.
- Pascal Van Hentenryck, Agostino Cortesi and Baudouin Le Charlier. "Type analysis of prolog using type graphs". In: *The Journal of Logic Programming* 22.3 (Mar. 1995), pp. 179–209. DOI: 10.1016/0743-1066(94)00021-w.
- Bertrand Jeannet. "Some Experience on the Software Engineering of Abstract Interpretation Tools". In: *Electronic Notes in Theoretical Computer Science* 267.2 (Oct. 2010), pp. 29–42. DOI: 10.1016/j.entcs.2010.09.016.
- Xavier Leroy and François Pessaux. "Type-based analysis of uncaught exceptions". In: *ACM Trans. Program. Lang. Syst.* 22.2 (2000), pp. 340–377. DOI: 10.1145/349214.349230.

- Benoît Montagu and Thomas P. Jensen. "Stable Relations and Abstract Interpretation of Higher-order Programs". In: *Proc. ACM Program. Lang.* 4.ICFP (2020), 119:1–119:30. DOI: 10.1145/3409001.
- Benoît Montagu and Thomas P. Jensen. "Trace-based control-flow analysis". In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 20211*. Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 482–496. DOI: 10.1145/3453483.3454057.
- Helmut Seidl and Ralf Vogler. "Three Improvements to the Top-Down Solver". In: *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*. Ed. by David Sabel and Peter Thiemann. ACM, 2018, 21:1–21:14. DOI: 10.1145/3236950.3236967.

- Olin Shivers. “The Semantics of Scheme Control-Flow Analysis”. In: *Proceedings of the 1991 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. PEPM ’91. New York, NY, USA: Association for Computing Machinery, 1991, pp. 190–198. ISBN: 0897914333. DOI: 10.1145/115865.115884.
- Paulo Emílio de Vilhena, François Pottier and Jacques-Henri Jourdan. “Spy game: verifying a local generic solver in Iris”. In: *Proc. ACM Program. Lang.* 4.POPL (2020), 33:1–33:28. DOI: 10.1145/3371101.
- Vesal Vojdani et al. “Static race detection for device drivers: the Goblint approach”. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016*. ACM, 2016, pp. 391–402. DOI: 10.1145/2970276.2970337.