Automated Differential Testing for OCaml Modules

Ernest Ng Advised by **Harry Goldstein** & **Benjamin C. Pierce**

Penn

Key Points

- The OCaml module system allows for representation independence, where the same ***interface*** (or ***module signature***) can admit different implementations. However, checking whether two modules are ***observationally equivalent*** requires significant programmer effort!

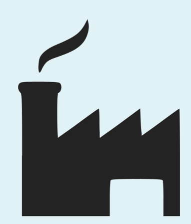
- We present **Mica**, a tool that automates differential testing for two OCaml modules implementing the same signature. Mica does this by *automatically* producing property-based testing (PBT) code specialized to the signature!

1. Property-Based Testing

(e.g. Haskell QuickCheck)

Write a ***property*** — executable spec describing desired behavior

Generate ***random inputs***



$\longrightarrow x_1\ x_2\ \ldots$

Test if random inputs satisfy property

$\forall x.\ P(x)$

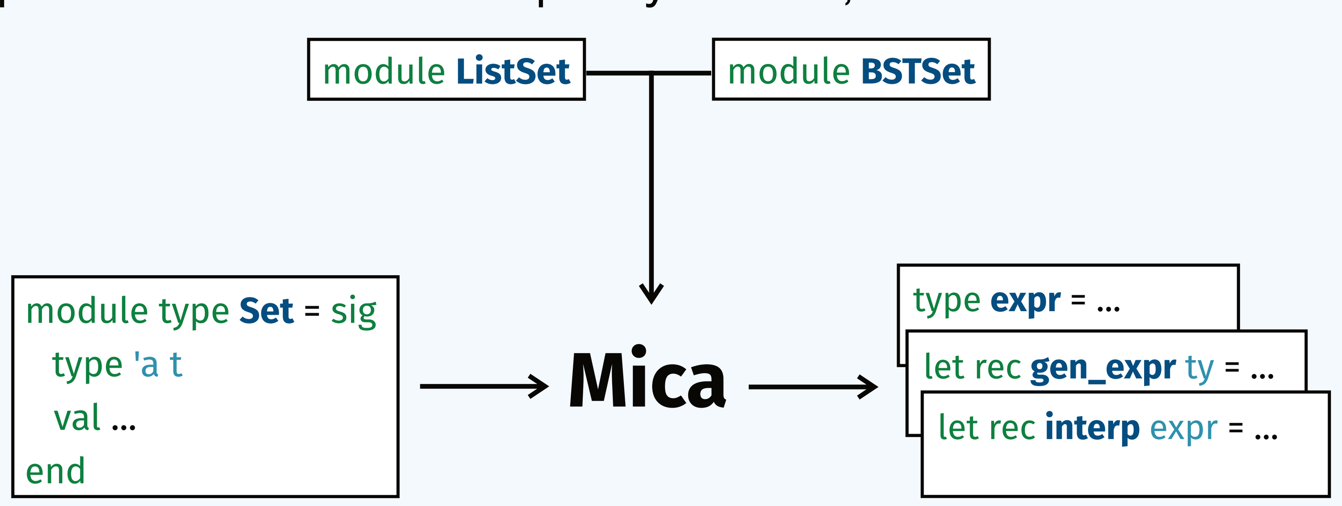
2. Motivating Example

Consider two implementations of finite sets that use BSTs & lists respectively:

module type **Set**	module **BSTSet**	module **ListSet**
{ 1 , 4 , 9 , ... }	4 / 1 9 /	[1 ; 4 ; 9 ; ...]

Mica tests these two modules for observational equivalence as follows:

1. Mica parses the signature, *automatically* producing datatype definitions & PBT functions specialized to the signature.
2. Mica then generates random *symbolic expressions* that correspond to invocations of the module functions.
3. Mica evaluates these expressions over the two modules, checking that the modules produce equivalent values. If a discrepancy is found, Mica warns the user!



References

[1] John Hughes. 2016. Experiences with QuickCheck: Testing the Hard Stuff and Staying Sane. Vol. 9600. 169–186.

[2] Jan Midtgaard. 2020. A Simple State-Machine Framework for Property-Based Testing in OCaml. In OCaml Workshop 2021.

[3] François Pottier. 2021. Strong automated testing of OCaml libraries. In *Journées Francophones des Langages Applicatifs*.

3. Auto-Generated Code

Mica *automatically* produces type & function definitions, specialized to the interface under test:

3.1 Symbolic Expressions

Each function in the **Set** interface from §2 corresponds to a constructor of the **expr** datatype with the same name, arity & argument types:

```
module type Set = sig                        
  type 'a t                                  type expr =
  val empty : 'a t                  ⟷        | Empty
  val is_empty : 'a t → bool        ⟷        | Is_Empty of expr
  val add : int → 'a t → 'a t       ⟷        | Add of int * expr
  val union : 'a t → 'a t → 'a t    ⟷        | Union of expr * expr | ...
end
```

3.2 Types & Values

Based on the return type of functions in the module signature, Mica generates datatypes representing the possible concrete types & values that **expr**s can return:

```
type ty = Bool | Int | T           type value =
                                     | ValBool of bool
                                     | ValInt of int
                                     | ValT of int M.t
```

T ≈ abstract type in module

M = module under test

3.3 QuickCheck Generator

`val gen_expr : ty → expr Generator.t`

`gen_expr τ` generates random ***well-typed*** symbolic expressions of type τ

`Union (Add 2 Empty) Empty` √ `Is_Empty (Size Empty)` X

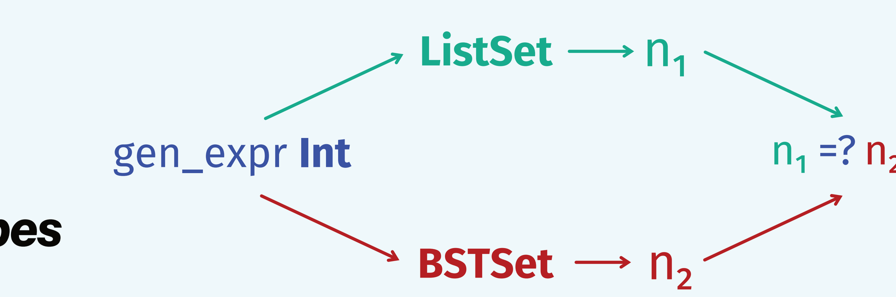
3.4 Interpreter

`val interp : expr → value`

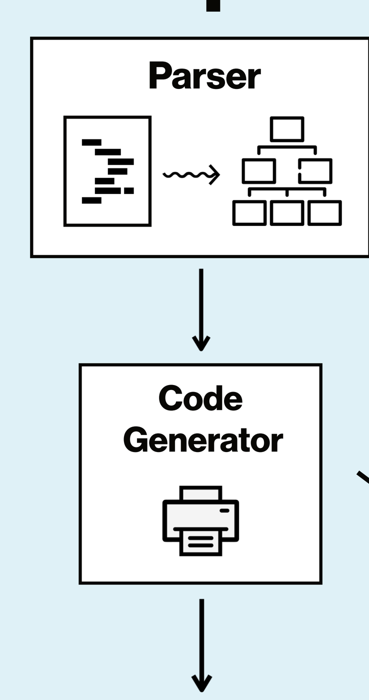
Interprets **expr**s over a module, returning corresponding **value**s

Top-level behavior:

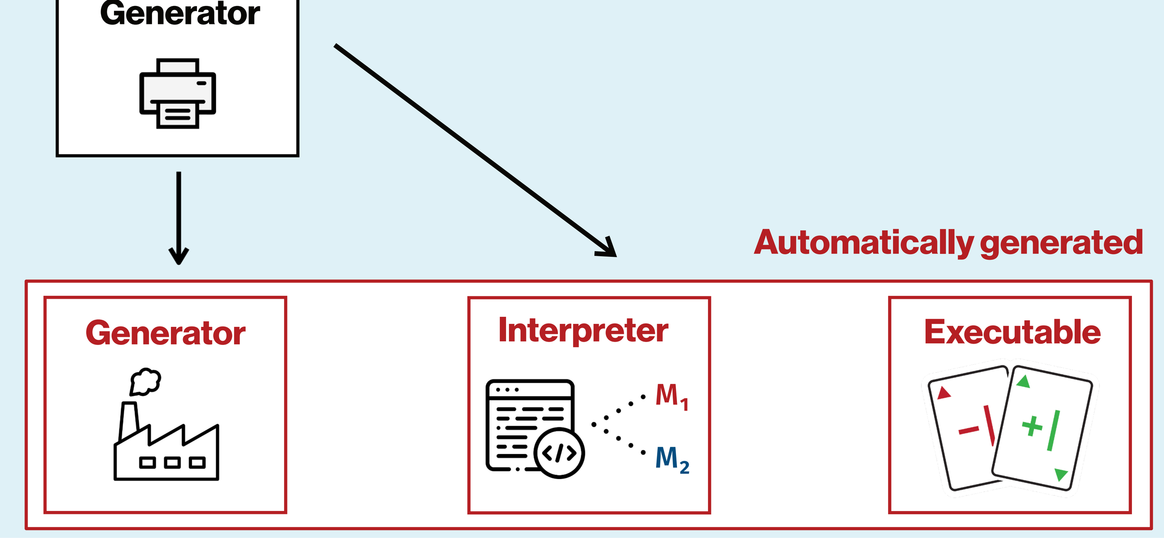
Mica compares the **value** of interpreted **expr**s at ***concrete types*** (e.g. `int`, but not `'a t`)



4. Implementation



Mica parses an OCaml module signature into an AST, using the type information in the AST to produce PBT code via Jane Street's Core.QuickCheck library.



5. Case Studies

(more examples on GitHub)

Regex matchers	**Finite Sets**	**Polynomials**
(Brzozowski derivatives & DFAs)	(Lists & BSTs)	(Horner schema & list folds)
$r ::= \varnothing \mid \epsilon \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$		$\sum_{i=1}^{n} c_i \cdot x^i$

15 manually inserted bugs caught across all modules

6. Future Work

- **Encode dependencies** in the generated symbolic expressions
- Support **imperative code**
- **Shrinker** for symbolic expressions

7. Related Work

- QuviQ QuickCheck (Hughes 2016)
- QCSTM (Midtgaard 2021)
- Monolith (Pottier 2021)

Mica furthers existing work in the differential & PBT literature by:

- *Automatically* deriving specialized testing code, obviating the need for users to learn specialized DSLs
- Supporting binary operations on abstract types in modules

ICFP SEATTLE 2023

GitHub: ngernest/mica

ngernest@seas.upenn.edu