ROTOR: First Steps Towards a Refactoring Tool for OCaml

Reuben N. S. Rowe
Simon Thompson
University of Kent, Canterbury

r.n.s.rowe@kent.ac.uk
s.j.thompson@kent.ac.uk

OCaml Users and Developers Workshop, Oxford, UK
Friday 8th September 2017
What is ROTOR?
What is Rotor?

Reliable
What is Rotor?

R eliable

O caml-based
What is Rotor?

Reliable
Caml-based
Tool for
What is ROTOR?

R eliable
O Caml-based
T ool for
O Caml
What is Rotor?

Reliable

Caml-based

Tool for

Caml

Refactoring
What is Rotor?

• Written in OCaml itself
What is ROTOR?

- Written in OCaml itself
  - allows re-use of the existing compiler infrastructure
What is Rotor?

- Written in OCaml itself
  - allows re-use of the existing compiler infrastructure

- Provide evidence that the result is correct (Future work)
What is Rotor?

- Written in OCaml itself
  - allows re-use of the existing compiler infrastructure

- Provide evidence that the result is correct (Future work)

- A prototype refactoring tool for OCaml (4.04.x) programs
What is ROTOR?

- Written in OCaml itself
  - allows re-use of the existing compiler infrastructure

- Provide evidence that the result is correct (Future work)

- A prototype refactoring tool for OCaml (4.04.x) programs
  - currently implements renaming for value bindings
What is ROTOR?

• Written in OCaml itself
  • allows re-use of the existing compiler infrastructure

• Provide evidence that the result is correct (Future work)

• A prototype refactoring tool for OCaml (4.04.x) programs
  • currently implements renaming for value bindings

• Designed with extensibility in mind
What is Rotor?

- Written in OCaml itself
  - allows re-use of the existing compiler infrastructure

- Provide evidence that the result is correct (Future work)

- A prototype refactoring tool for OCaml (4.04.x) programs
  - currently implements renaming for value bindings

- Designed with extensibility in mind
  - write new refactorings and ‘plug them in’ easily
Renaming Value Bindings: Rebinding

```
src/foo.ml:

let f = ... 
let f = ... 
...
...

src/bar.ml:

open Foo

...
... f ...
```

Foo.f ↦ g
Renaming Value Bindings: Rebinding

```
src/foo.ml:
  :
  let f = ...
  let g = ...
  :
  ...
  ...

src/bar.ml:
  open Foo
  :
  :
  ...
  ...
```

\[\text{Foo.f} \leftrightarrow \text{g}\]
Renaming Value Bindings: Rebinding

\[\text{src/foo.ml:}\]
\[
\begin{align*}
\text{let } & g = \ldots \\
\text{let } & f = \ldots \\
\text{let } & f = \ldots \\
& \vdots \\
& \ldots \ldots 
\end{align*}
\]

\[\text{src/bar.ml:}\]
\[
\begin{align*}
\text{open } & \text{Foo} \\
& \vdots \\
& \ldots f \ldots g \ldots
\end{align*}
\]

\[\text{Foo.f} \leftrightarrow g\]
src/foo.ml:

```ml
let g = ...
let f = ...
let g = ...

... g ...
```

src/bar.ml:

```ml
open Foo

... g ...
```

Foo.f ↔ g
src/foo.ml:

```ocaml
let f = ...
```

Foo.f \(\mapsto\) g
src/foo.ml:

```ml
type t = { f : ... ; ... }
let f = ...
  :
... { f; ... } : t ...
```

src/bar.ml:

```ml
open Foo
let map ~f xs = ...
  :
... map ~f ['a';'b';'c'] ...
```
Renaming Value Bindings: Punning

src/foo.ml:

```ml
type t = { f : ... ; ... }
let g = ...
  ...
  ... { f=g; ... } : t ...
```

src/bar.ml:

```ml
open Foo
let map ~f xs = ...
  ...
  ... map ~f:g ['a';'b';'c'] ...
```
Renaming Value Bindings: include

src/foo.ml:

let f = ...
::
... f ...

src/bar.ml:

include Foo
::

src/baz.ml:

::

... Bar.f ...
Renaming Value Bindings: include

src/foo.ml:

```ml
let g = ...
... g ...
```

src/bar.ml:

```ml
include Foo
... Bar.g ...
```
Renaming Value Bindings: include

src/foo.ml:
  let g = ...
  ...
  ... g ...

src/bar.ml:
  include Foo
  ...

src/baz.ml:
  ...
  ... Bar.g ...

Foo.f ↦ g
Bar.f ↦ g
Renaming Value Bindings: Module Signatures

src/foo.ml:
  let f = ...

src/bar.ml:
  include Foo

Foo.f \mapsto g

Bar.f \mapsto g
Renaming Value Bindings: Module Signatures

src/foo.ml:
  let f = ...

src/bar.ml:
  include Foo

src/bar.mli:
  include Sig.S

src/sig.ml:
  module type S = sig val f : ... end
Renaming Value Bindings: Module Signatures

src/foo.ml:
  let f = ...

src/bar.ml:
  include Foo

src/bar.mli:
  include Sig.S

src/sig.ml:
  module type S = sig val f : ... end
Renaming Value Bindings: Module Signatures

src/foo.ml:

```ml
let f = ...
```

Foo.f ⟷ g

src/bar.ml:

```ml
include Foo
```

Bar.f ⟷ g

src/bar.mli:

```ml
include Sig.S
```

src/sig.ml:

```ml
module type S = sig val f : ... end
```

Sig.S.f ⟷ g

src/baz.ml:

```ml
module M : Sig.S = struct let f = ... end
```
Renaming Value Bindings: Module Signatures

src/foo.ml:

```ml
let g = ...
```

Foo.f → g

src/bar.ml:

```ml
include Foo
```

Bar.f → g

src/bar.mli:

```ml
include Sig.S
```

Sig.S.f → g

src/sig.ml:

```ml
module type S = sig val g : ... end
```

src/baz.ml:

```ml
module M : Sig.S = struct let g = ... end
```
Renaming Value Bindings: Module Signatures

src/foo.ml:
let g = ...

src/bar.ml:
include Foo

src/bar.mli:
include Sig.S

src/sig.ml:
module type S = sig val g : ... end

src/baz.ml:
module M : Sig.S = struct let g = ... end
Visitors

Path_visitors
Longident_visitors

Types_visitors
Parsetree_visitors
Typedtree_visitors
**ROTOR: Architectural Overview**

### Compiler-libs
- Path
- Longident
- Types
- Parsetree
- Typedtree

### Visitors
- Path_visitors
- Longident_visitors
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors
**ROTOR: Architectural Overview**

### Compiler-libs
- Path
- Longident
- Types
- Parsetree
- Typetree

### Visitors
- Path_visitors
- Longident_visitors
- Types_visitors
- Parsetree_visitors
- Typetree_visitors
ROTOR: Architectural Overview

Compiler-libs
- Path
- Longident
  - :
- Types
- Parsetree
- Typedtree

Visitors
- Path_visitors
  - :
- Longident_visitors
  - :
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors

Language
- Elements
  - Identifier
  - View
  - Deps
Adi su yere na za na cewa a cikin zan iya da顏sasa.

1. **Compiler-libs**
   - Path
   - Longident
   - Types
   - Parsetree
   - Typedtree

2. **Visitors**
   - Path_visitors
   - Longident_visitors
   - Types_visitors
   - Parsetree_visitors
   - Typedtree_visitors

3. **Language**
   - Elements
   - Identifier
   - View
   - Deps

4. **Infrastructure**
   - Fileinfos
   - Sourcefile
   - Codebase
   - Buildenv
Rotor: Architectural Overview

**Compiler-libs**
- Path
- Longident
- Types
- Parsetree
- Typedtree

**Visitors**
- Path_visitors
- Longident_visitors
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors

**Language**
- Elements
- Identifier
- View
- Deps

**Infrastructure**
- Fileinfos
- Sourcefile
- Codebase
- Buildenv

**Refactoring**
- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors
ROTOR: Architectural Overview

Compiler-libs
- Path
- Longident
- Types
- Parsetree
- Typedtree
- Path_visitors
- Longident_visitors
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors

Refactoring
- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors

Language Elements
- Identifier
- View
- Deps

Infrastructure
- Fileinfos
- Sourcefile
- Codebase
- Buildenv

Refactoring
- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors

module Replacement :
  sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end

module Refactoring :
  sig
    module Repr :
      sig
        type t
        module Set : Set.S with type elt = t
      end
      module type S = sig
        val repr : Repr.t
        val get_deps : Sourcefile.t -> Repr.Set.t
        val process_file : Sourcefile.t -> Replacement.Set.t
        val kernel : Codebase.t -> Fileinfos.t list
      end
    end
  end

module Refactoring_lib :
  sig
    val of_repr : Refactoring.Repr.t -> (module Refactoring.S)
  end
module Replacement : sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end

module Refactoring : sig
  module Repr :
    sig
      type t
      module Set : Set.S with type elt = t
    end
  module type S = sig
    val repr : Repr.t
    val get_deps : Sourcefile.t -> Repr.Set.t
    val process_file : Sourcefile.t -> Replacement.Set.t
    val kernel : Codebase.t -> Fileinfos.t list
  end
end

module Refactoring_lib : sig
  val of_repr : Refactoring.Repr.t -> Refactoring.S
end
**ROTOR: Architectural Overview**

**Compiler-libs**
- Path
- Longident
- Types
- Parsetree
- Typedtree

**Refactoring**
- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors

```ocaml
module Replacement : sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end

module Refactoring : sig

end

module Refactoring_lib :
  val of_repr : Refactoring.Repr.t -> (module Refactoring.S)

end
```
module Replacement : sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end

module Refactoring : sig
  module Repr : sig
    type t
    module Set : Set.S with type elt = t
  end

  module type S = sig
    val repr : Repr.t
    val get_deps : Sourcefile.t -> Repr.Set.t
    val process_file : Sourcefile.t -> Replacement.Set.t
    val kernel : Codebase.t -> Fileinfos.t list
  end
end

module Refactoring_lib : sig
  val of_repr : Refactoring.Repr.t -> (module Refactoring.S)
end
module Replacement : sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end

module Refactoring : sig
  module Repr : sig
    type t
    module Set : Set.S with type elt = t
  end
  module type S = sig
    val repr : Repr.t
    val get_deps : Sourcefile.t -> Repr.Set.t
    val process_file : Sourcefile.t -> Replacement.Set.t
    val kernel : Codebase.t -> Fileinfos.t list
  end
end


**Compiler-libs**

- Path
- Longident
- Types
- Parsetree
- Typedtree

**Visitors**

- Path_visitors
- Longident_visitors
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors

**Language**

- Elements
- Identifier
- View
- Deps

**Infrastructure**

- Fileinfos
- Sourcefile
- Codebase
- Buildenv

**Refactoring**

- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors

**Rename**

- Rename_code
- Rename_val_impl
- Rename_val_intf
**ROTOR: Architectural Overview**

**Compiler-libs**
- Path
- Longident
- :
- Types
- Parsetree
- Typedtree

**Visitors**
- Path_visitors
- Longident_visitors
- :
- Types_visitors
- Parsetree_visitors
- Typedtree_visitors

**Language**
- Elements
- Identifier
- View
- Deps

**Infrastructure**
- Fileinfos
- Sourcefile
- Codebase
- Buildenv

**Refactoring**
- Replacement
- Refactoring
- Refactoring_lib
- Refactoring_utils
- Refactoring_visitors

**Rename**
- Rename_code
- Rename_val_impl
- Rename_val_intf

**Driver**
- Configuration
- Frontend
- Main
Using the Visitors PPX

type foo = Null | Foo of int * bar
and bar = { name : string; baz : foo }
[@@deriving visitors { variety = "map" } ]
Using the Visitors PPX

type foo = Null | Foo of int * bar
     and bar = { name : string; baz : foo }
[@@deriving visitors { variety = "map" } ]

class virtual ['self] map = object (self : 'self)
    inherit [_] Visitors_runtime.iter
    method visitFoo v =
    
    match v with
    | Null -> Null
    | Foo (v1, v2) ->
        let v1' = self#visit_int v1 in
        let v2' = self#visit_bar v2 in
        Foo (v1', v2')
    method visitBar v =
        let v1 = self#visit_string v.name in
        let v2 = self#visitFoo v.baz in
        { name = v1; baz = v2; }
    end
Using the Visitors PPX

```ocaml
type foo = Null | Foo of int * bar
and bar = { name : string; baz : foo }
[@@deriving visitors { variety = "map" } ]

let double = object (self)
  inherit [ ] map
  method! visit_int v =
    2 * v
end ;;

let v = Foo (3, { name = "Outer"; baz = Foo (5, { name = "Inner"; baz = Null }); } ) ;;

double#visit_foo v ;;
- : foo = Foo (6, { name = "Outer"; baz = Foo (10, { name = "Inner"; baz = Null }); } )
```

Using the Visitors PPX

```
#use compiler-libs ;;

module Typedtree_visitors = struct
  type tt_structure = Typedtree.structure = ...
  and tt_structure_item_desc = Typedtree.structure_item_desc =
    | Tstr_value of ...
    | Tstr_type of ...
  :

end
```
Using the Visitors PPX

#use compiler-libs ;;

module Typedtree_visitors = struct
    type tt_structure = Typedtree.structure = ...
    and tt_structure_item_desc = Typedtree.structure_item_desc =
        | Tstr_value of ...
        | Tstr_type of ...
    ...
    [@@deriving visitors { variety = "iter" },
    visitors { variety = "map" },
    visitors { variety = "reduce" }
    ]
end
#use compiler-libs ;;

module Types_visitors = struct ... end
module Parsetree_visitors = struct ... end

module Typedtree_visitors = struct
  type tt_structure = Typedtree.structure = ... 
  and tt_structure_item_desc = Typedtree.structure_item_desc =
    | Tstr_value of ...
    | Tstr_type of ...
  ...
  [@@deriving visitors { variety = "iter",
    ancestors = [
      "Types_visitors.iter" ; "Parsetree_visitors.iter" ] },
    visitors { variety = "map", ancestors = ... },
    visitors { variety = "reduce", ancestors = ... } ]
end
Demo!
Experimental Testbed: Jane Street’s core Library

- The **core** library + its dependencies:
  - `core_kernel`, `base`, `stdio`, `sexplib`, `ppx_`...
  - ~900 source files, ~80 libraries
Experimental Testbed: Jane Street’s core Library

- The **core** library + its dependencies:
  - `core_kernel`, `base`, `stdio`, `sexplib`, `ppx_`...
  - ~900 source files, ~80 libraries

- Collected ~4000 different identifiers of locally defined bindings (~1000 are operators)
Experimental Testbed: Jane Street’s core Library

- The **core** library + its dependencies:
  - `core_kernel`, `base`, `stdio`, `sexplib`, `ppx_`...
  - ~900 source files, ~80 libraries

- Collected ~4000 different identifiers of locally defined bindings (~1000 are operators)

<table>
<thead>
<tr>
<th>Refactoring Failed (exception)</th>
<th>Rebuild Failed</th>
<th>Rebuild Succeeded</th>
</tr>
</thead>
<tbody>
<tr>
<td>821 (27%)</td>
<td>1462 (47%)</td>
<td>786 (26%)</td>
</tr>
</tbody>
</table>
### Experimental Testbed: Jane Street’s core Library

#### Rebuild Succeeded

<table>
<thead>
<tr>
<th></th>
<th>Files</th>
<th>Hunks</th>
<th>Avg. Hunks/File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>50</td>
<td>128</td>
<td>5.7</td>
</tr>
<tr>
<td>Mean</td>
<td>4.8</td>
<td>7.1</td>
<td>1.3</td>
</tr>
<tr>
<td>Mode</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Rebuild Failed

<table>
<thead>
<tr>
<th></th>
<th>Files</th>
<th>Hunks</th>
<th>Avg. Hunks/File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>11</td>
<td>369</td>
<td>18</td>
</tr>
<tr>
<td>Mean</td>
<td>5.5</td>
<td>11.0</td>
<td>1.5</td>
</tr>
<tr>
<td>Mode</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Next Steps ...

- Increase coverage of the renaming refactoring:
Next Steps ...

- Increase coverage of the renaming refactoring:
  - Fix bugs identified via the testbed
Next Steps ...

- Increase coverage of the renaming refactoring:
  - Fix bugs identified via the testbed
  - Handle functors

- Extend renaming to cover other language elements (e.g. modules, types, classes, etc.)

- Integrate the tool with editors/workflows (e.g. emacs, version control)

- Incorporate formal correctness guarantees
  - Make use of the CakeML HOL formalisation
Next Steps ...

• Increase coverage of the renaming refactoring:
  • Fix bugs identified via the testbed
  • Handle functors
  • Extend renaming to cover other language elements (e.g. modules, types, classes, etc.)
Next Steps ...

- Increase coverage of the renaming refactoring:
  - Fix bugs identified via the testbed
  - Handle functors
  - Extend renaming to cover other language elements (e.g. modules, types, classes, etc.)

- Integrate the tool with editors/workflows (e.g. emacs, version control)
Next Steps ...

- Increase coverage of the renaming refactoring:
  - Fix bugs identified via the testbed
  - Handle functors
  - Extend renaming to cover other language elements (e.g. modules, types, classes, etc.)

- Integrate the tool with editors/workflows (e.g. emacs, version control)

- Incorporate formal correctness guarantees
Next Steps ...

- Increase coverage of the renaming refactoring:
  - Fix bugs identified via the testbed
  - Handle functors
  - Extend renaming to cover other language elements (e.g. modules, types, classes, etc.)

- Integrate the tool with editors/workflows (e.g. emacs, version control)

- Incorporate formal correctness guarantees
  - Make use of the CakeML HOL formalisation
gitlab.com/trustworthy-refactoring/refactorer

www.cs.kent.ac.uk/projects/trustworthy-refactoring/

cakeml.org