Rotor: First Steps Towards a Refactoring Tool for OCaml

Reuben N. S. Rowe r.n.s.rowe@kent.ac.uk
Simon Thompson s.j.thompson@kent.ac.uk

University of Kent, Canterbury

OCaml Users and Developers Workshop, Oxford, UK
Friday 8th September 2017

What is Rotor?

What is ROTOR?

R eliable

What is ROTOR?

R eliable

O Caml-based

What is Rotor?

R eliable

O Caml-based

T ool for

What is Rotor?

R eliable

O Caml-based

T ool for

O Caml

What is ROTOR?

R eliable

O Caml-based

T ool for

O Caml

R efactoring

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 = ...
    ⋮
  ... f ...

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

Foo.f ↦ g

Renaming Value Bindings: Rebinding

Foo.f ↦ g

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

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

Renaming Value Bindings: Rebinding

Foo.f ↦ g

src/foo.ml:

```
let g = ...
let f = ...
let f = ...
  ⋮
...  ...
```

src/bar.ml:

```
open Foo
  ⋮
... f ... g ...
```

Renaming Value Bindings: Rebinding

src/foo.ml:

```
let g = ...
let f = ...
let g = ...
    ⋮
... g ...
```

src/bar.ml:

```
open Foo
    ⋮
... g ... g ...
```

Foo.f ↦ g

✗

Renaming Value Bindings: Punning

```
src/foo.ml:
  type t = { f : ... ; ... }
  let f = ...
    ⋮
  ... { f; ... } : t ...
```

Foo.f ↦ g

Renaming Value Bindings: Punning

Foo.f ↦ g

```
src/foo.ml:
  type t = { f : ... ; ... }
  let f = ...
    ⋮
  ... { f; ... } : t ...

src/bar.ml:
  open Foo
  let map ~f xs = ...
    ⋮
  ... map ~f ['a';'b';'c'] ...
```

Renaming Value Bindings: Punning

Foo.f ↦ g

```
src/foo.ml:
  type t = { f : ... ; ... }
  let g = ...
    ⋮
  ... { f=g; ... } : t ...

src/bar.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 ...
```

Foo.f ↦ g

Renaming Value Bindings: include

Foo.f ↦ g

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

src/bar.ml:
  include Foo
    ⋮

src/baz.ml:
    ⋮
  ... Bar.g ...
```

Renaming Value Bindings: include

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

Foo.f ↦ g

```
src/bar.ml:
  include Foo
    ⋮
```

Bar.f ↦ g

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

Renaming Value Bindings: Module Signatures

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

Foo.f ↦ g

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

Bar.f ↦ g

Renaming Value Bindings: Module Signatures

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

src/bar.ml:                               Bar.f ↦ g
  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:                                  Foo.f ↦ g
  let f = ...

src/bar.ml:                                  Bar.f ↦ g
  include Foo

src/bar.mli:
  include Sig.S

src/sig.ml:                                  Sig.S.f ↦ g
  module type S = sig val f : ... end
```

Renaming Value Bindings: Module Signatures

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

src/bar.ml:                                   Bar.f ↦ g
  include Foo

src/bar.mli:
  include Sig.S

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

src/baz.ml:
  module M : Sig.S = struct let f = ... end
```

Renaming Value Bindings: Module Signatures

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

src/bar.ml:                         Bar.f ↦ g
  include Foo

src/bar.mli:
  include Sig.S

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

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

Renaming Value Bindings: Module Signatures

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

src/bar.ml:                                   Bar.f ↦ g
  include Foo

src/bar.mli:
  include Sig.S

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

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

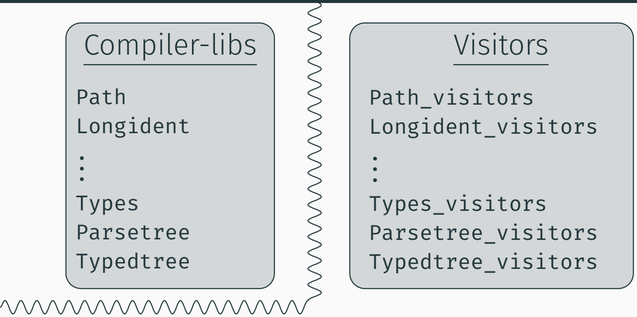
Rotor: Architectural Overview

Rotor: Architectural Overview

Visitors

```
Path_visitors
Longident_visitors
⋮
Types_visitors
Parsetree_visitors
Typedtree_visitors
```

ROTOR: Architectural Overview



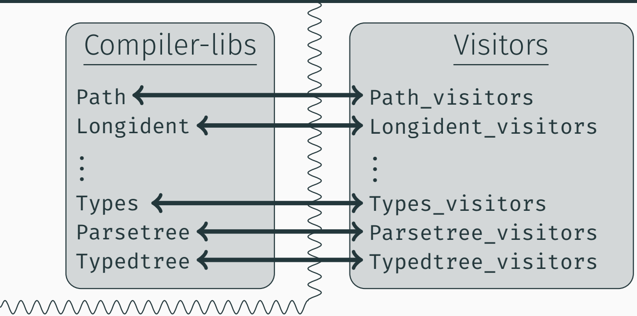
Compiler-libs

```
Path
Longident
⋮
Types
Parsetree
Typedtree
```

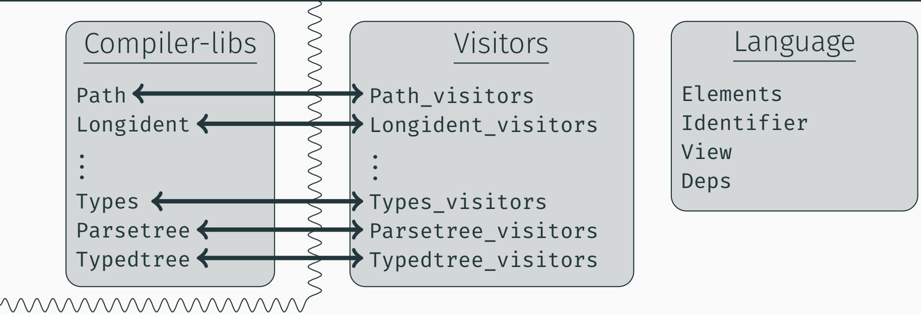
Visitors

```
Path_visitors
Longident_visitors
⋮
Types_visitors
Parsetree_visitors
Typedtree_visitors
```

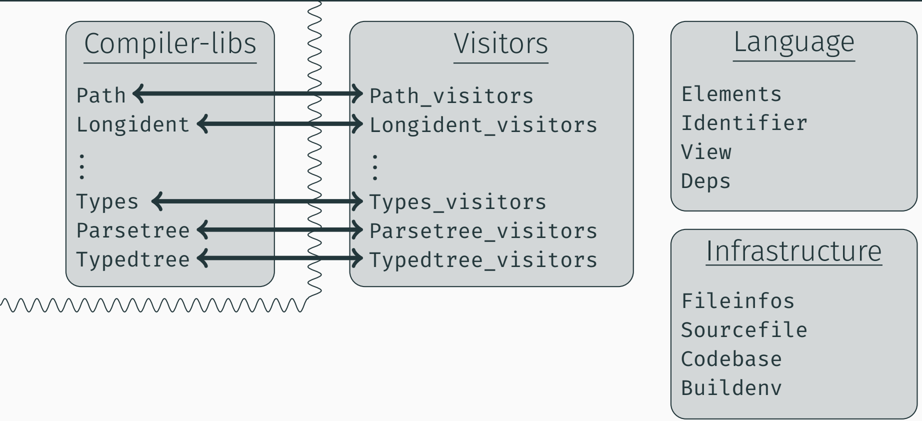
ROTOR: Architectual Overview



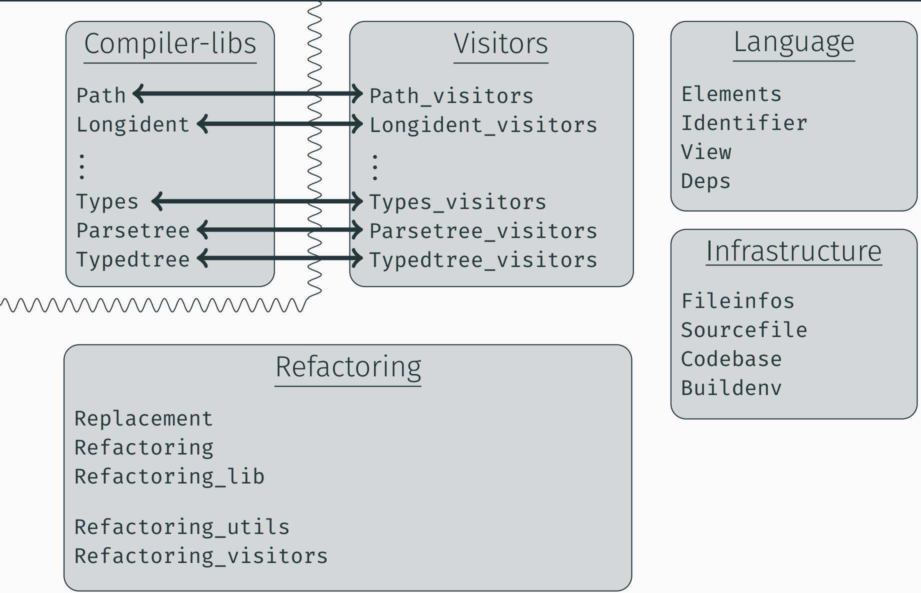
ROTOR: Architectural Overview



Rotor: Architectural Overview



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

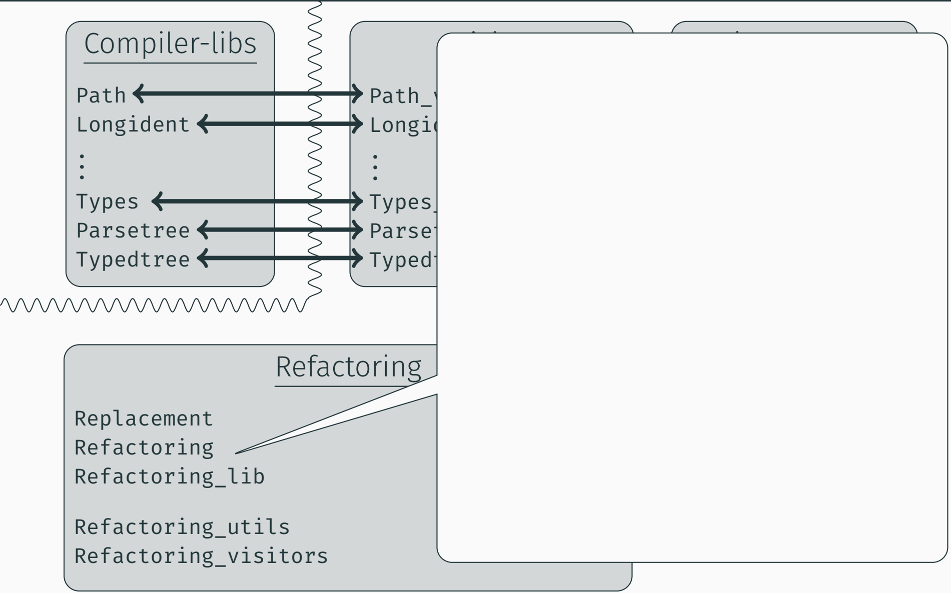
Fileinfos
Sourcefile
Codebase
Buildenv

Refactoring

Replacement
Refactoring
Refactoring_lib

Refactoring_utils
Refactoring_visitors

ROTOR: Architectural Overview



Compiler-libs

Path

Longident

⋮

Types

Parsetree

Typedtree

Path_

Longi

⋮

Types_

Parse

Typed

Refactoring

Replacement

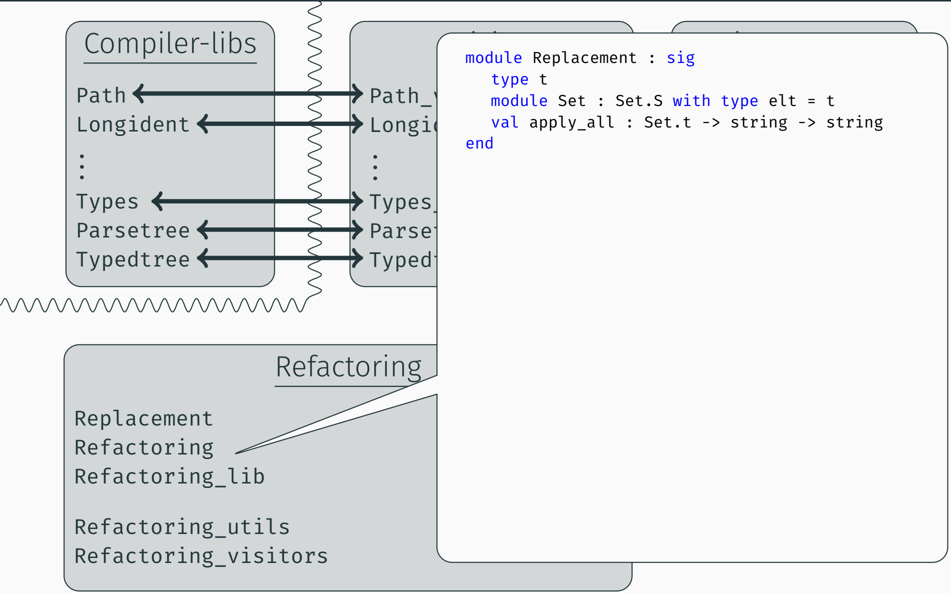
Refactoring

Refactoring_lib

Refactoring_utils

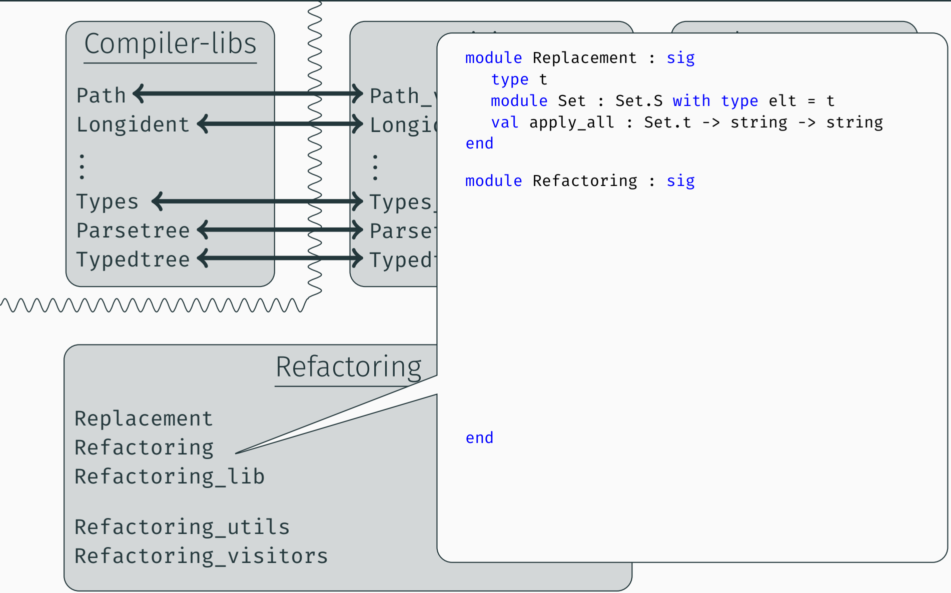
Refactoring_visitors

ROTOR: Architectural Overview



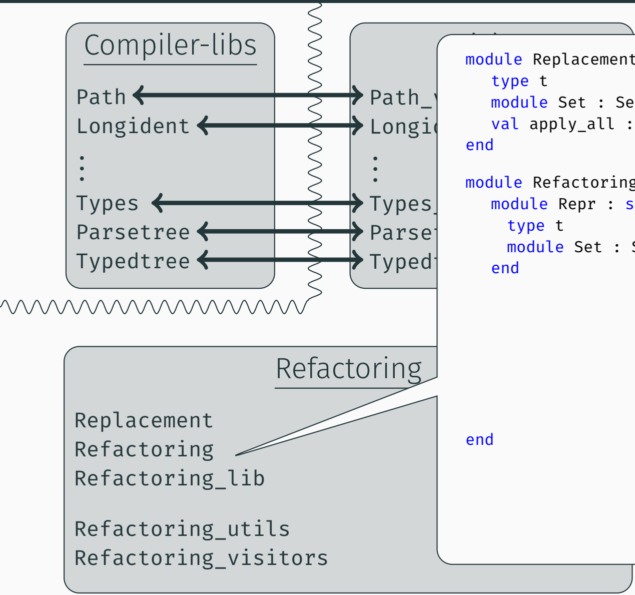
```
module Replacement : sig
  type t
  module Set : Set.S with type elt = t
  val apply_all : Set.t -> string -> string
end
```

ROTOR: Architectural Overview



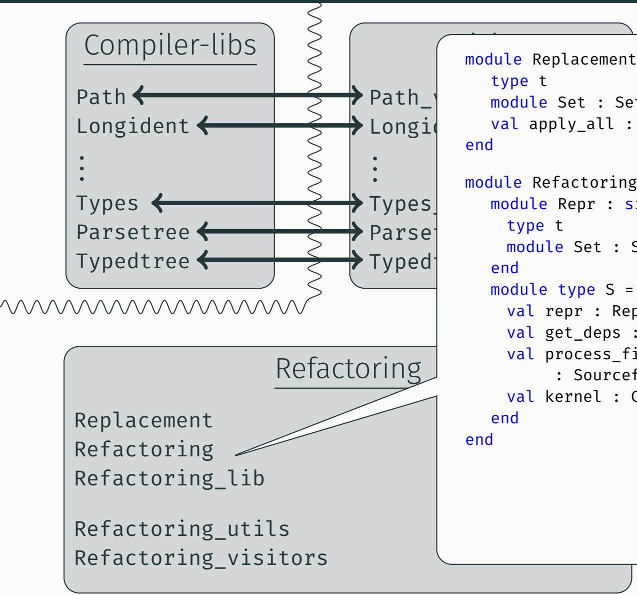
```
module Replacement : sig
   type t
   module Set : Set.S with type elt = t
   val apply_all : Set.t -> string -> string
end

module Refactoring : sig









end
```

ROTOR: Architectural Overview



```
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









end
```

ROTOR: Architectural Overview



Compiler-libs

Path
Longident
⋮
Types
Parsetree
Typedtree

Path_
Longi
⋮
Types_
Parse
Typed

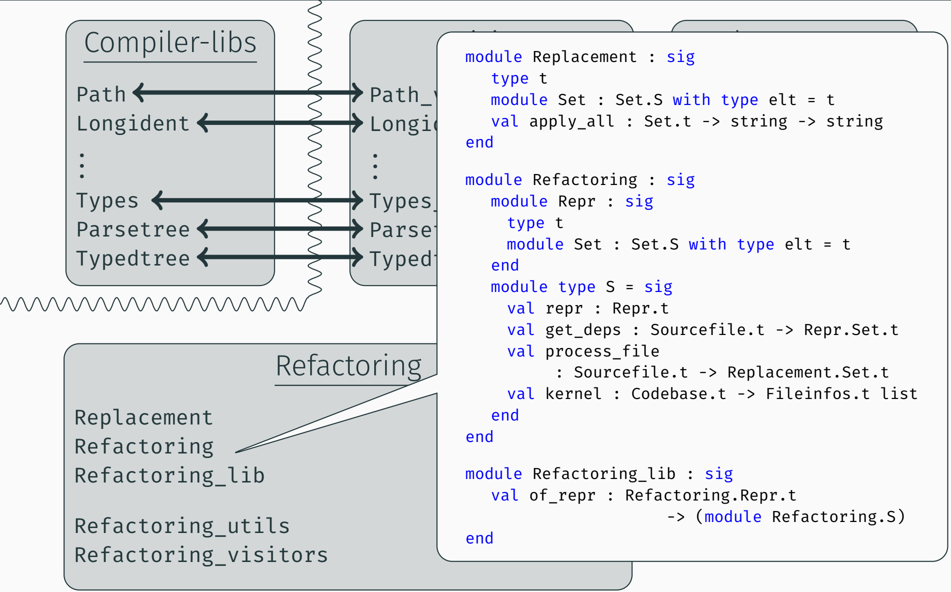
Refactoring

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
```

ROTOR: Architectural Overview



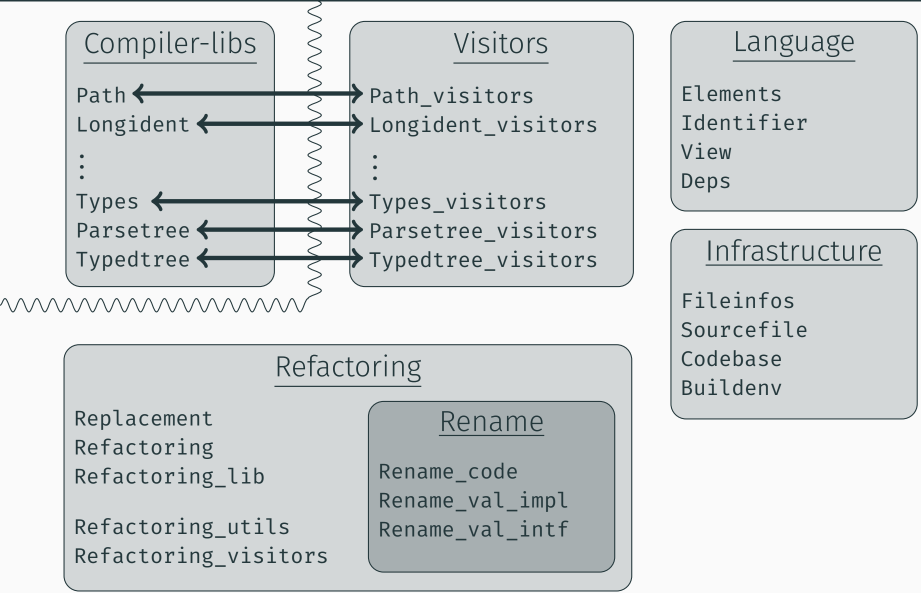
Refactoring

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

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

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

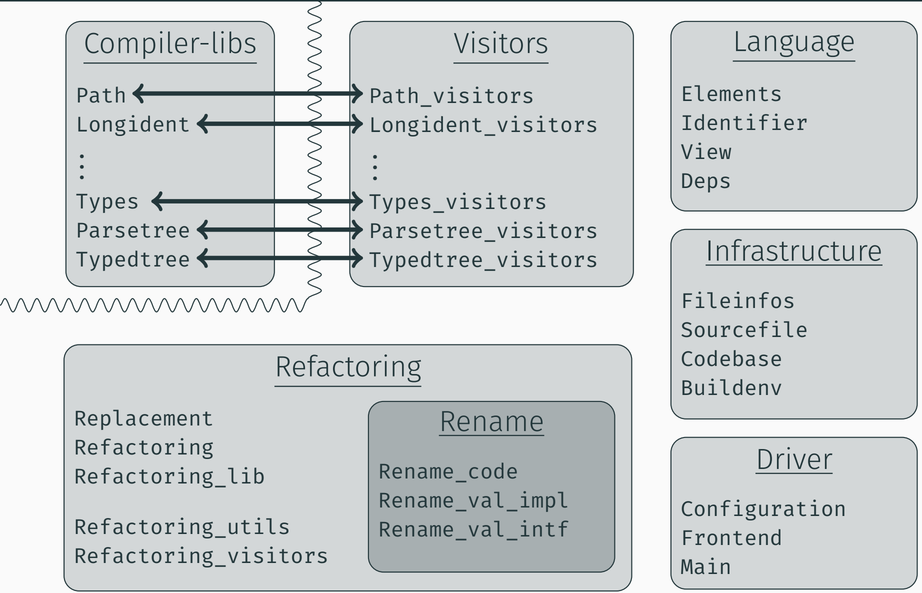
Replacement
Refactoring
Refactoring_lib

Refactoring_utils
Refactoring_visitors

Rename

Rename_code
Rename_val_impl
Rename_val_intf

Rotor: Architectual 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 visit_foo 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 visit_bar v =
    let v1 = self#visit_string v.name in
    let v2 = self#visit_foo v.baz in
    { name = v1; baz = v2; }
end
```

Using the Visitors PPX

```
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
```

Using the Visitors PPX

```
#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)

Refactoring Failed (exception)	Rebuild Failed	Rebuild Succeeded
821 (27%)	1462 (47%)	786 (26%)

Experimental Testbed: Jane Street's `core` Library

Rebuild Succeeded

	Files	Hunks	Avg. Hunks/File
Max	50	128	5.7
Mean	4.8	7.1	1.3
Mode	3	3	1

Rebuild Failed

	Files	Hunks	Avg. Hunks/File
Max	11	369	18
Mean	5.5	11.0	1.5
Mode	2	2	1

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

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
A Verified Implementation of ML

cakeml.org