

# High-level error messages for modules through diffing

---

Florian ANGELETTI

Inria

`florian.angeletti@inria.fr`

Gabriel RADANNE

Inria

`gabriel.radanne@inria.fr`

```
module String_map = Map.Make(String)(String)
```

```
module String_map = Map.Make(String)(String)
```

**Error:** This module is not a functor; it has type

```
sig
```

```
  type key = String.t
```

```
  type 'a t = 'a Map.Make(String).t
```

```
  val empty : 'a t
```

```
  val is_empty : 'a t -> bool
```

```
  val mem : key -> 'a t -> bool
```

```
  val add : key -> 'a -> 'a t -> 'a t
```

```
  val update : key -> ('a option -> 'a option) -> 'a t -> 'a t
```

```
  val singleton : key -> 'a -> 'a t
```

```
  ...(elided for the slide)
```

```
end
```

```
module String_map = Map.Make(String)(String)
```

**Error:** The functor application is ill-typed.

These arguments:

```
String String
```

do not match these parameters:

```
functor (Ord : Map.OrderedType) -> ...
```

1. Module String matches the expected module type Map.OrderedType

2. The following extra argument is provided

```
String : ...(elided)
```

**Theory:** Most complex part of the language

**Practice:** Shallow hierarchy of submodules, a handful of functors with few arguments

## A typical example

A common example of functors (such as Ocamlgraph)

```
module Graph(Vertex:VERTEX)(Edge:EDGE) = struct ... end
```

## Common errors

- Unnecessary argument

```
module G = Graph(Label)(Vertex)(Edge)
```

- Forgotten argument:

```
module G = Graph(Edge)
```

- Wrong argument:

```
module G = Graph(Label)(Edge)
```

All those errors are frequent during refactorings.

Classical error messages consider only the last alternative:

```
module G = Graph(Label)(Vertex)(Edge)
```

**Error:** Signature mismatch:

Modules do not match:

```
sig type t = string end
```

is not included in

```
VERTEX
```

The value `label' is required but not provided

The value `create' is required but not provided

The type `label' is required but not provided

The value `equal' is required but not provided

The value `hash' is required but not provided

The value `compare' is required but not provided



## A little bit of perspective

For errors, we need to keep the *multi-application* context

Consider the arguments as two lists:

```
Graph(Label)(Vertex)(Edge)
```

```
module Graph(Vertex: VERTEX)(Edge: EDGE) = ...
```

use diffing between lists or strings: addition, deletion, or change.

## Unnecessary argument

```
module G=Graph(Label)(Vertex)(Edge)
```

**Error:** The functor application is ill-typed.

These arguments:

```
Label Vertex Edge
```

do not match these parameters:

```
functor (Vertex : VERTEX) (Edge : EDGE) -> ...
```

1. The following extra argument is provided  
Label : sig type t = string end
2. Module Vertex matches the expected module type VERTEX
3. Module Edge matches the expected module type EDGE

## Missing argument

```
module G=Graph(Edge)
```

**Error:** The functor application is ill-typed.

These arguments:

Edge

do not match these parameters:

```
functor (Vertex : VERTEX) (Edge : EDGE) -> ...
```

1. An argument appears to be missing with module type VERTEX
2. Module Edge matches the expected module type EDGE

## Wrong argument

```
module G=Graph(Label)(Edge)
```

**Error:** The functor application is ill-typed.

These arguments:

**Label** Edge

do not match these parameters:

functor (**Vertex** : VERTEX) (Edge : EDGE) -> ...

1. Modules do not match:

Label : sig type t = string end

is not included in

VERTEX

...(elided for the slide)

2. Module Edge matches the expected module type EDGE

**How does this work?**

Error classes seen this far: Addition, Deletion, Change

Those are classical errors in string diffing

String diffing is everywhere: spellchecking, fuzzy search, control version, DNA sequence matching, ...

## String diffing – details

Typical operations:

- Addition: A B C D
- Deletion: A B ~~C~~ D
- Change: A B ~~F~~<sup>C</sup> D
- Swap: A B C D  
↔

Well-studied topic:

- many distances: Levenstein, LCS, Hamming, ...
- many algorithms: Wagner-Fischer, Myers diff, Hirschberg's algorithm, ...

## String diffing – details

Typical operations:

- **Addition:** A B C D
- **Deletion:** A B ~~C~~ D
- **Change:** A B ~~F~~<sup>C</sup> D
- **Swap:** A B C D  
                  ↔

Well-studied topic:

- many distances: Levenstein, LCS, Hamming, ...
- many algorithms: Wagner-Fischer, Myers diff, Hirschberg's algorithm, ...



## String diffing – details

Typical operations:

- **Addition:** A B C D
- **Deletion:** A B ~~C~~ D
- **Change:** A B ~~F~~<sup>C</sup> D
- **Swap:** A B C D  
↔

Well-studied topic:

- many distances: **Levenstein**, LCS, Hamming, ...
- many algorithms: **Wagner-Fischer**, Myers diff, Hirschberg's algorithm, ...

# Wagner-Fischer illustrated

		A	B	X	D	E
0	0	<b>1</b>	2	3	4	5
B	1	2	<b>1</b>	2	3	3
C	2	3	2	<b>2</b>	3	3
D	3	4	3	3	<b>2</b>	<b>3</b>

The diagram shows a dynamic programming table for the Wagner-Fischer algorithm. The columns represent characters A, B, X, D, E and the rows represent characters B, C, D. The top-left cell (0,0) is 0. Red arrows indicate the path of minimum edit distance: from (0,0) to (0,1), then to (1,1), then to (2,2), then to (3,2), and finally to (3,3). The values along this path are 1, 1, 2, 2, and 3 respectively.

Dynamically-sized variant of the Wagner-Fischer algorithm for computing distance with addition-deletion-change

- Argument comparison: OCaml typechecker comparison used as a grey box
- Complexity:  $O(\max(|arguments|, |parameters|)^2)$
- Merged in the OCaml compiler!

See <https://github.com/ocaml/ocaml/pull/9331>

**Does it work in practice?**

## Unreasonably complicated error

```
module A = Seq module A' = Complex
module B = Option module B' = Bigarray
module C = Result module C' = Bool
module D = Char module D' = Uchar
module E = Sys module E' = List
module F = Bytes module F' = Map.Make
module G = String module G' = Unit
module H = Marshal module H' = Obj
```

## Unreasonably complicated error

```
module Little_functor =  
  (A: module type of A) (B: module type of B)  
  (C: module type of C) (D: module type of D)  
  (E: module type of E) (F: module type of F)  
  (G: module type of G) (H: module type of H)  
= struct end
```

# Unreasonably complicated error

```
module E = Little_functor(A)(A)(B)(C')(E)(F')(G)(H)(H')
```

These arguments:

```
A A B C' E F' G H H'
```

do not match these parameters:

```
functor (A : ...) (B : ...) (C : ... (C)) (D : ... (D)) (E : ...)
(F : ... (F)) (G : ...) (H : ...) -> ...
```

1. Module A matches the expected module type
2. The following extra argument is provided: ...(elided)
3. Module B matches the expected module type
4. Modules do not match: C' : ...(elided)
5. An argument appears to be missing ...(elided)
6. Module E matches the expected module type
7. Modules do not match: F' ...(elided)
8. Module G matches the expected module type
9. Module H matches the expected module type
10. The following extra argument is provided H' : ...(elided)

## Dependent functors?

```
module type f = functor
  (X:sig type a type b end)
  (Y:sig type foo = Foo of X.a end)
  (Z:sig type bar = Bar of X.b end)
-> sig end
```

```
module F = functor
  (X:sig type a type b end)
  (Z:sig type bar = Bar of X.b end)
-> struct end
```

```
(* Check (F : f) *)
```



## Dependent functors?

**Error:** Signature mismatch:

Modules do not match:

```
functor (X : $S1) (Z : $S3) -> ...
```

is not included in

```
functor (B : $T1) (Y : $T2) (Z : $T3) -> ...
```

1. Module types \$S1 and \$T1 match
2. An argument appears to be missing with module type

```
$T2 = sig type foo = Foo of X.a end
```

3. Module types \$S3 and \$T3 match

## Dynamically-sized?

Functors are complicated, they can be variadic in OCaml:

```
module F(X:sig
  module type t
  module M:t
end) = X.M
module Ft = struct
  module M = F
  module type t = module type of F
end
```

```
F(Ft)(Ft)(Ft)(Ft)...
```

Our modified Wagner-Fischer algorithm handles those cases (and yet always terminates).

## What about swaps?

Hard to present well in text output.

We may consider a subcase where all arguments are here:

```
Graph (Edge) (Vertex)
```

Hypothetical error message:

```
The functor Arguments appears to be in the wrong order.
```

```
Did you mean Graph(Vertex)(Edge)?
```

## A question of performances?

- Quadratic module comparisons
- Errors are for human. Human are slow.
- Which effect matters?

## A not that small functor

- A 26 argument functor, with all stdlib modules involved

These arguments:

A A B C' E F' G H H' I J L M N' O O' P R S T' U' V W W X Y Z Z

do not match these parameters:

```
functor (A : ...) (B : ...) (C : ... (C)) (D : ... (D)) (E : ...)
(F : ... (F)) (G : ...) (H : ...) (I : ...) (J : ...) (K : ... (K))
(L : ...) (M : ...) (N : ... (N)) (O : ...) (P : ...) (Q : ... (Q))
(R : ...) (S : ...) (T : ... (T)) (U : ... (U)) (V : ...) (W : ...)
(X : ...) (Y : ...) (Z : ...) -> ...
```

- 100 ms with ocamlpt

If you have existing code (before the 27 August 2020), with functors with more than ten arguments, we are offering a bounty of one drink.<sup>1</sup>

---

<sup>1</sup>Mirage-generated code does not count

If you have existing code (before the 27 August 2020), with functors with more than ten arguments, we are offering a bounty of one drink.<sup>1</sup>

**CLAIMED**

---

<sup>1</sup>Mirage-generated code does not count

# Conclusion and future works

## Take-away

- Ideas do not have to be complicated, and can come from anywhere!  
⇒ Here, the idea came from DOM-manipulation in Web programming
- It can take a long time to merge stuff in the compiler. . .

## Applications: The diffing hammer

- Diffing for functor application and inclusion with Wagner-Fischer  
**Done!**
- Signature difference (ground work done, UI to be done)
- Sum types  
**Done by Florian, Pull Request incoming**
- Moving to other structural types (in OCaml: objects, polymorphic variants)



**The End**