The Memory Behavior Of OCaml Programs

OUD 2012

Çagdas Bozman[1,2,3]
Thomas Gazagnaire[1]
Fabrice Le Fessant[2]
Michel Mauny[3]

OCamlPro[1] | INRIA[2] | ENSTA-ParisTech[3]

14 Sept. 2012

Memory Problems

- What ?
  - Study the memory behavior of OCaml programs
  - Memory profiling tools
- Why ?
  - To decrease memory footprint
  - To fix memory leaks
  - To spend less time in memory management

Using Too Much Memory

- Applications that use too much memory
- Impact: applications with big memory usage (symbolic computation tools, etc.)
- Which type would you choose between t_1 and t_2?

```
type t1 = {
  f1:  float;
  f2:  float;
}
```

```
type t2 = (float * float)
```

Using Too Much Memory

- Applications that use too much memory
- Impact: applications with big memory usage (symbolic computation tools, etc.)
- Which type would you choose between t_1 and t_2? t_1

```
type t1 = {
  f1:  float; 8/8 bytes
  f2:  float; 8/8 bytes
}
 ⇒16/16 bytes
```

```
type t2 = (float * float)
⇒ 32/48 bytes
```

- Causes: bad data representation, bad use of collection, etc.

Memory Leaks

- Applications that fail to free the memory they have used
- Add values to a collection and never remove them

```
try
    Hashtbl.add tbl x y;
    do_something_that_may_fail tbl;
    Hashtbl.remove tbl x
with _ → ()
```

- Equivalent of malloc() without free() in C

Cost Of Memory Management

- Spending too much time in memory management
-

```
let rec x = ref 1.0 in
for i=1 to 1_000_000_000 do
    x := 1.  *.  !x
done;
!x *.  1.0
```

  - with `*.  1.0`: 1.8s
  - without: 3.6s
- Need tools to understand when and why memory is allocated

Consequences

- Best case:
  slow the application (swapping, garbage collection time)
- Worst case:
  run out of memory $\Rightarrow$ crash

Tooling

- What kind of tools we are developing to help to understand memory usage ?
  - Allocation Profiler
  - Region Inference
  - Snapshot Memory Profiler
  - Continuous Memory Profiler

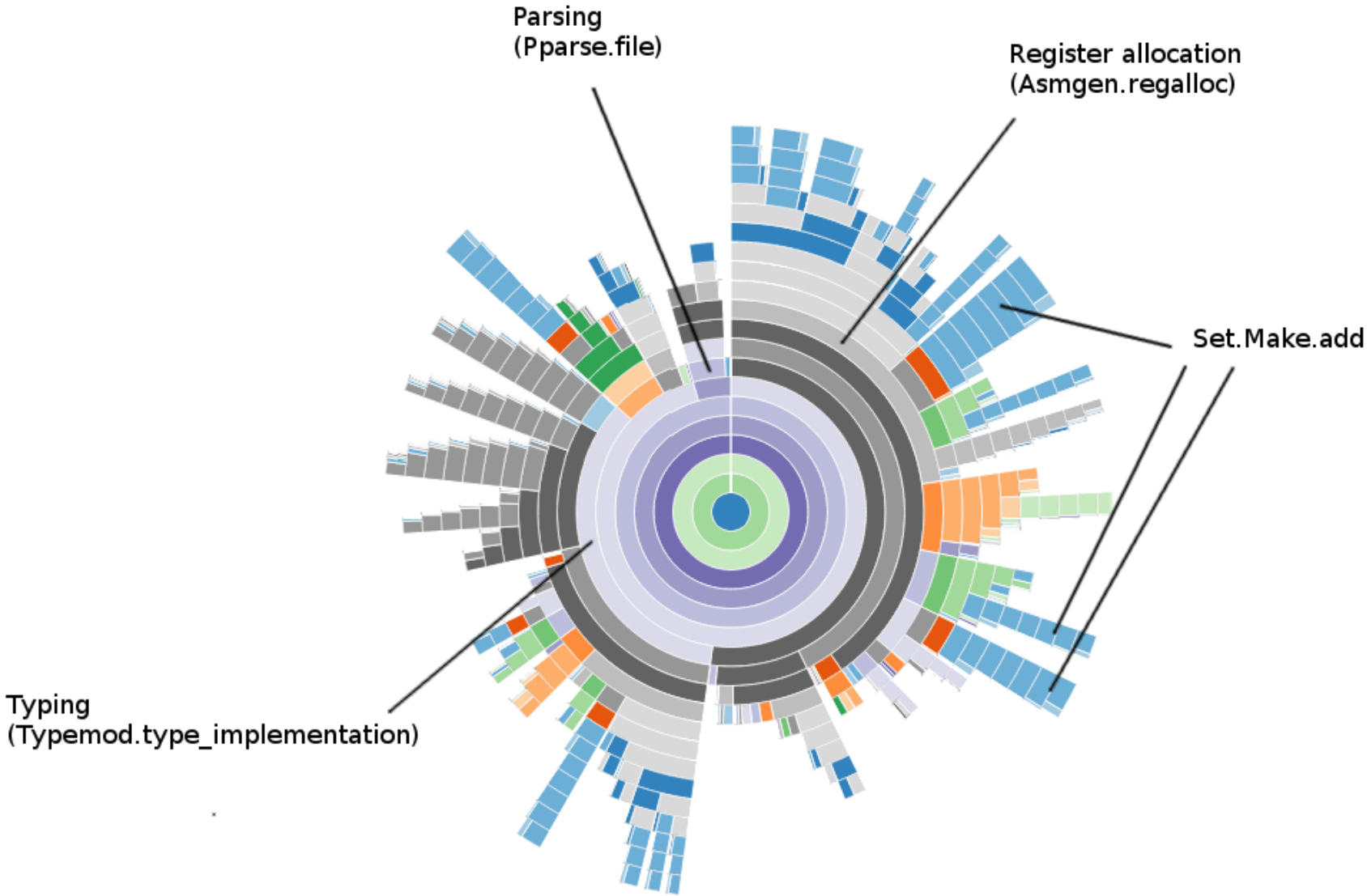
Allocation Profiling

- Profile where an application allocates memory
- Inspired from the Poor's man method
  - record information on the stack state periodically after some allocation events.
- Display allocation hotspots
- Approximated call graph, weighted by the number of bytes allocated
- Need only to be linked with a modified runtime (libasmrun.a), to save backtraces at sampling points
- Use OCAMLRUNPARAM to set sampling rate

Allocation Profiling - Demo

1. Link `ocamlopt.opt` with the modified runtime
2. Configure sampling rate
   `export OCAMLRUNPARAM=M=4k`
3. Run your program → save stack backtraces in `mem.out`
   `ocamlopt.opt -c -I */ typing/*.ml`
4. Generate a `json` file
   `ocp-memprof -space-prof mem.out -format json`
5. Here is the graph (next slide)

Allocation Profiling - `ocamlopt.opt`



Region Inference

- Approximating values lifetime using Region Inference [Tofte and Talpin]
  - every value is assigned to a region
  - region is a kind of lexical scope
- Annotations help us to understand values interferences
  $\Rightarrow$ easier tracking of unwanted values in a region

Region Inference - Example

```
let tbl1 = Hashtbl.create 3
let tbl2 = Hashtbl.create 3

let x11 = (1, 1)
let x12 = (1, 2)
let x21 = (2, 1)
let x22 = (2, 2)

let add_tbl1 = Hashtbl.replace tbl1
let add_tbl2 = Hashtbl.replace tbl2

let f_tbl1 cond =
  if cond then
    add_tbl1 x11 11
  else
    add_tbl2 x12 12

let f_tbl2 cond =
  if cond then
    add_tbl2 x21 21
  else
    add_tbl2 x22 22


let _ =
  add_tbl1 x11 11;
  add_tbl1 x12 12;
  add_tbl2 x21 21;
  add_tbl2 x22 22
```

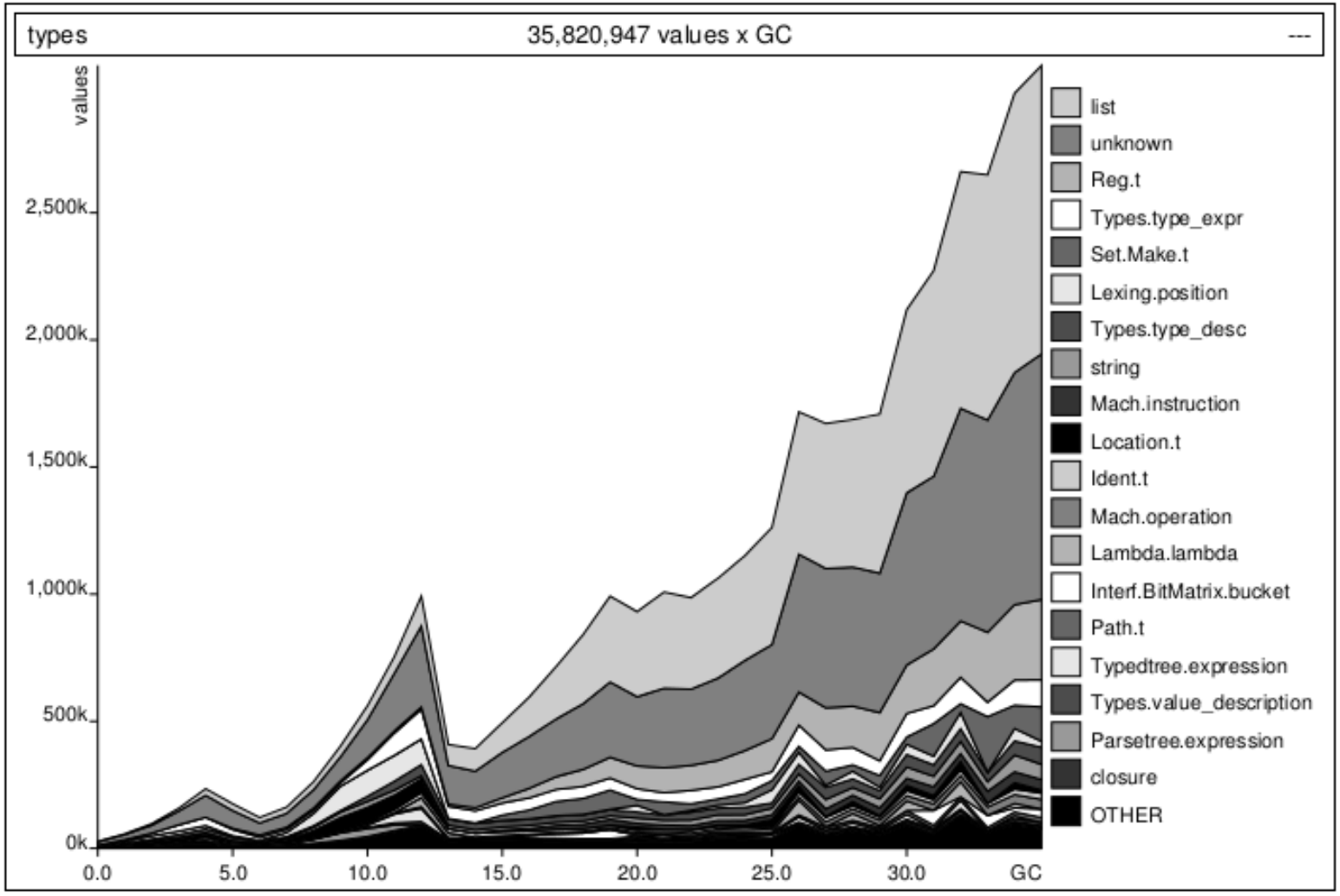
```
let tbl1 = Hashtbl.create 3
let tbl2 = Hashtbl.create 3

let x11 = (1, 1)
let x12 = (1, 2)
let x21 = (2, 1)
let x22 = (2, 2)

let add_tbl1 = Hashtbl.replace tbl1
let add_tbl2 = Hashtbl.replace tbl2

let f_tbl1 cond =
  if cond then
    add_tbl1 x11 11
  else
    add_tbl1 x12 12

let f_tbl2 cond =
  if cond then
    add_tbl2 x21 21
  else
    add_tbl2 x22 22


let _ =
  add_tbl1 x11 11;
  add_tbl1 x12 12;
  add_tbl2 x21 21;
  add_tbl2 x22 22
```

Snapshot Profiling

- Work in progress
- Detailed liveness informations
- We have:
  - Dumps of memory graph
- We want:
  - Recover types and names: what type/value eats all my memory ?

Snapshot Profiling - Graph

```
$ ocamlopt.opt -c -I utils -I parsing -I typing typing/*.ml
```



Continuous Profiling

- Work in progress
- Global memory behavior
- We have:
  - Log allocation and GC events
- We want:
  - Extract useful things from raw data

Conclusion

Tools:

- Allocation Profiler: allocations hotspots
- Region Inference: values interaction
- Snapshot Memory Profiler: detailed liveness information
- Continuous Memory Profiler: global memory behavior

Futur Works:

- Display tools
- Recover more types (snapshot)
- Recover memory blocks lifetime (continuous)