Profiling the Memory Usage of OCaml Applications without Changing their Behavior

Çağdaş Bozman — Fabrice Le Fessant — Michel Mauny

cagdas.bozman@ocamlpro.com — fabrice.le_fessant@inria.fr — michel.mauny@ensta.fr

Context

- **OCaml:** strong type system, no need for runtime type information to perform computation
  Concise and Fast ⇒ improve your coding efficiency while producing code with higher quality
- **Consequences**: Memory representation much more compact ⇒ better performance
- **Problem** : Absence of runtime type information becomes a problem to recover information from heap-allocated values

Objectives of the Work

- Recover useful information in the heap: type and location of allocated blocks
- Profiling memory behavior without impacting performance nor adding memory overhead
- Find symptoms of memory issues (leaks, for instance)

General Idea

The OCaml Language

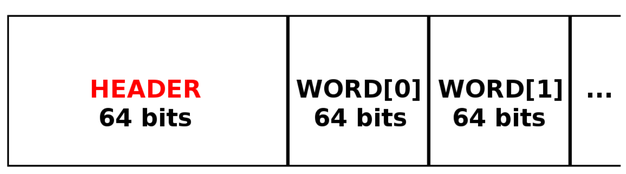
- High-Level programming language
- *Automatic memory management*: generational and incremental garbage collector
  - ⊕ no more explicit allocation/deallocation
  - ⊕ avoid a whole class of bugs such as dangling pointers, double frees, etc.
  - ⊖ some value might never be reclaimed (e.g. when stored in a global hash-table)
  - ⊖ takes time to decide which memory cells to free, so when the memory pressure increases, automatic memory management can become a performance problem

Idea

- Propagate the source location of the allocation site until the memory blocks handled by the garbage collector
- Allow the programmer to dump the memory with these locations
- Provide tools that read and recover the location and type information:
  - profiling with snapshots: post-mortem inspection of memory dumps
  - continuous profiling: connecting memory blocks to their allocation point, during the execution of the application

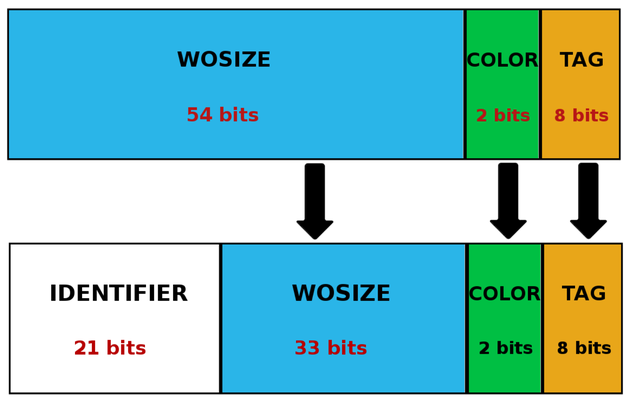
Hiding information in the blocks' header

Generation of a random identifier by the OCaml compiler to allow us later to associate a source location to each memory block in the heap.



An OCaml block consists of:

- a *header* (32/64 bits)
- an array of values or pointers to values (32/64 bits for each)



A closer look to the header:

- *Wosize* represents the number of words in the block
- *Color* is used by the Garbage Collector
- *Tag* is used to encode the minimal type information for standard operations (e.g. pattern-matching, polymorphic comparisons, etc.)

Generating Identifiers

Identifiers are generated by a simple calculation: a hash of the location and the digest of the module name.

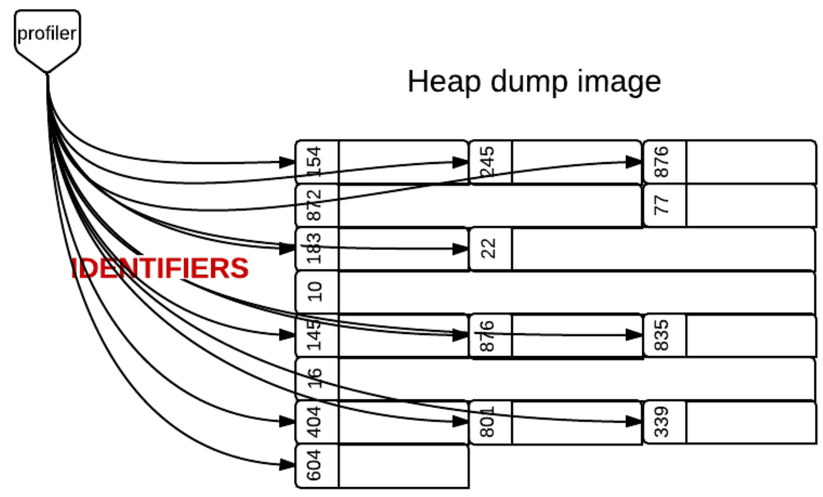
Storing Identifiers

After the generation and association of the pair identifier-location, we need to store identifiers in the heap.

Advantages:

- no space overhead
- no performance overhead

The Profiler



1. Dump one or more images of the heap
2. Get identifiers from the header of each block
3. Find locations from identifiers
4. Recover the types of the blocks
5. Build the graph

Conclusion and Perspective

- **Conclusion**
  - set of patches to the compiler and runtime
  - dump heap images into a journal
  - better understanding of the memory usage

- **Future Work**
  - tracking other interesting information
    - values from minor to major generation heap
    - values which stay alive after a number of major collections
    - lifetime of values
  - **exploit this information for detecting and explaining memory behaviors**