Study of OCaml programs' memory behavior

Çagdas Bozman
ENSTA-ParisTech & OCamlPro
cagdas.bozman@ocamlpro.com

Thomas Gazagnaire
OCamlPro
thomas.gazagnaire@ocamlpro.com

Fabrice Le Fessant
INRIA & OCamlPro
fabrice.le_fessant@inria.fr

Michel Mauny
ENSTA-ParisTech
michel.mauny@ensta.fr

Abstract

In this paper, we present a preliminary work on new memory profiling tool and others, to help us to understand memory behavior.

1 Introduction

OCaml is a strongly typed functional programming language with automatic memory management. Indeed, an incremental garbage collector (GC) frees the programmer from manually dealing with memory allocation/deallocation. Among the benefits of automatic memory management, it ensures that a large class of bugs can never happen, such as dangling pointers, double frees, etc. However, it also comes with some drawbacks: some values might never be reclaimed by the garbage collector, or collection might be delayed too much. As a consequence, memory pressure can increase, sometimes to have no more available free memory, or the garbage collector itself can consume a lot of resources to decide which memory cells to free, and both problems can lead to performance issues.

Thus, there is a need for tools to help developers profile the memory usage of their OCaml programs, but almost nothing is currently available.

In Section 2, we present a work (in progress) about `regions-infer`, a new tool based on region inference, which computes lifetime of values in a program.

In Section 3, we show `alloc-prof`, a tool that allows programmers to profile memory allocation and see hotspots in their programs.

Then, in Section 4, we describe `OCamlmemprof` which allows to understand the memory behavior by dumping memory information in a file.

Finally, in Section 5, we end up with a summary.

2 Region Inference

`region-infer` provides a simplified analysis of the lifetime of values in a program, based on region inference. Region inference was originally introduced by Tofte and Talpin [5, 6] for region based memory management, which is a compromise between manual and automatic memory management. With region based memory management, every value is assigned to a region, which is a collection of allocated blocks that will all be reclaimed together when execution leaves their region. Region inference is a static analysis technique that relieves the programmer from manually associating regions to objects. Tofte and Birkedal [4] explained that in some cases, the compiler is able to prove the absence of memory leaks or detect some kinds of memory leaks.

To verify this hypothesis, we have modi-

fied the OCaml code generator to infer regions and annotates values with them. These annotations help to understand which values are interfering (i.e. they are stored in the same region), and for how long. We then built a plugin in TypeRex [2] to display these regions in the source code, so that developers can easily see links and interactions between values in a program.

3 Memory allocator

`alloc-prof` allows developers to discover allocation hotspots in their programs. It is inspired from an email posted in the OCaml mailing-list, presenting the *poor man's profiler* [1, 7]. It is a modified OCaml runtime that saves in a journal the stack of the program every time a given amount of memory is allocated. The journals can then be displayed inside a simple web page written in js_of_ocaml [3], in order to display and explore the call graph (Figure 1), weighted by the number of allocation events.

Figure 1: Example of `alloc-prof` call graph

4 OCamlmemprof

`mem-prof` allows developers to understand how the memory of an OCaml application is used at any given time during the application execution. Two modes are available: (1) *continuous profiling*, and (2) *snapshot profiling*.

4.1 Continuous Profiling

In this mode, continuous profiling, the program is instrumented to store in each block some information about its type and its allocation point. During the execution, a journal stores allocation events and garbage collection events. The journal can then be read to display graphs showing the content of the heap at any given garbage collection event, as amounts of types or amounts of memory allocated by each function or module.

4.2 Snapshot Profiling

In this second mode, snapshot profiling, the program is not instrumented. Instead, a function is provided that dumps the content of memory in a file. This file is then analyzed to approximate the types of the memory blocks, and show how much memory is retained by every global value.

5 Conclusion

Using these tools, it is now possible to have a better idea of the memory behavior of OCaml programs. Once the profiling done, we need to understand results and then try to improve the user code by using some optimization techniques.

To summarize, all of the three tools modify the OCaml distribution. `mem-prof` and `alloc-prof` modify the OCaml runtime to dump allocation and deallocation events in a journal. `region-infer` modifies the OCaml compiler to compute regions after type inference, so that region information is available just after compilation.

We hope that these tools will help developers spot memory problems in their programs. Advanced users can use this information to improve their programs by reducing alloca-

tions in their programs' critical paths. For less advanced users, we plan to develop other tools that can provide advice on how to reduce the memory cost of a particular function or type.

References

[1] Poor man's profiler. http://poormansprofiler.org.

[2] OCamlPro. Typerex, a development environment for ocaml. http://typerex.org/.

[3] Ocsigen. Js_of_ocaml is a compiler from ocaml bytecode programs to javascrip. http://ocsigen.org/js_of_ocaml/manual/overview.

[4] M. Tofte and L. Birkedal. A region inference algorithm. *ACM Trans. Program. Lang. Syst.*, 20(4):724–767, July 1998.

[5] M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value &-calculus using a stack of regions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '94, pages 188–201, New York, NY, USA, 1994. ACM.

[6] M. Tofte and J.-P. Talpin. Region-based memory management. *Inf. Comput.*, 132:109–176, February 1997.

[7] ygrek. Poor man's allocation profiler. http://ygrek.org.ua/p/code/pmpa.