A WasmGC Backend for MicroHs

MARTIJN VOORWINDEN

Binaries produced by Haskell compilers such as GHC are often large, which can be a problem when targeting environments where programs need to be downloaded often, such as web applications, or environments that do not have much storage space, such as embedded systems. Currently, a large part of the binary size of a compiled program consists of its runtime system, especially of the garbage collector or memory manager.

Newer WebAssembly (Wasm) versions support garbage collection natively, which allows engines to reuse their built-in garbage collectors for programs compiled to Wasm, majorly reducing the size of produced binaries.

MicroHs is a lightweight Haskell compiler with one of its noticeable features being its small runtime system. Combined with its simple architecture (as compared to GHC) and relatively small codebase, it makes an ideal candidate for experiments with reducing binary sizes further by leveraging Wasm's native garbage collection support.

Previously, Apoorva Anand started work on a WasmGC backend and runtime system for MicroHs as part of his master's thesis. His work consists of three main parts: a WasmGC backend for the MicroHs compiler, a mostly handwritten runtime system to run SK combinator programs, and a DSL for generating combinator implementations for the runtime system. Although his work did not reach a state of being able to compile runnable programs, it showed promising results with regards to binary size reduction when comparing to binaries produced by the C backend of MicroHs and the existing Wasm backend through emscripten.

To build upon this work, we plan to undertake the following steps:

(1) Extend the existing DSL to generate more of the runtime system (preferably all of it) to make it easier to maintain, extend, and debug.
(2) Determine if the current MicroHs primitives and combinators are a good fit for the WasmGC backend, or if they should be reduced or extended to better align with WasmGC features.
(3) Determine how to best implement more complex language features such as IO in the WasmGC backend and runtime system. We expect our runtime system to differ from the C runtime system in this regard, as instead of relying on capabilities of the runtime system we can now request certain functions to be provided by the Wasm host environment.
(4) Implement missing features to be able to compile and run non-trivial programs, albeit for a limited subset of Haskell.
(5) Benchmark the produced binaries in terms of size, runtime, and memory usage against other backends of MicroHs.

We hope this project will be useful in showing the potential of leveraging WasmGC for reduced binary sizes and its accompanying performance impacts. Further benefits of this work could include the ability to restrict programs in what they can do, as certain capabilities are to be provided by the host environment, as well as the potential for easier interoperability with other programs through its use of Wasm as a compilation target.