A WasmGC Backend for MicroHs

MARTIJN VOORWINDEN, APOORVA ANAND, and MARCO VASSENA

Today, porting Haskell code to web browsers and cloud platforms is complicated. Approaches such as transpiling Haskell to JavaScript suffer from performance issues and lack of semantic similarity (purely functional vs imperative, lazy vs eager evaluation, etc.), while containerization solutions can be too heavyweight for resource-constrained environments and cannot simply run on the client side.

WebAssembly (Wasm) was designed to address issues, firstly by being a low-level efficient compilation target which does not come with all the overhead of a full heavily opinionated language such as JavaScript, and secondly by being a portable format that can run with a low footprint in a variety of environments including web browsers.

Traditional Wasm, however, did not natively support garbage collection, which forced languages to ship their own memory management systems, resulting in larger binary sizes and increased complexity. Since the last version of the WebAssembly specification, garbage collection support (WasmGC) has been added by leveraging the existing garbage collectors in VMs. This eliminates the need for applications to include their own memory management systems, potentially reducing binary sizes significantly.

In this project, we will be leveraging WasmGC to potentially decrease the binary size of Haskell programs compiled to WebAssembly. We will do this by extending a Haskell compiler with a WasmGC backend and runtime system. In particular, we will be using the MicroHs compiler, as its portable by design philosophy alligns well with the goals of Wasm, and its relatively smaller codebase and complexity than GHC makes it more suitable for a research project of this scale.

In previous work, we have built a WasmGC backend and a DSL that automates the implementation of SK combinator rewriting rules. Although this 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 that make use of the prelude or a subset thereof.
(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.