GHC's JavaScript Backend

Sylvain HENRY

INPUT | OUTPUT

GHC Workshop
7-9 June 2023

2023: Haskell in the browser

- Two new backends arrived at once in GHC 9.6
- WebAssembly backend
  - Cf yesterday's presentation by Cheng
- JavaScript backend
  - This presentation

Table of Contents

1. Motivation
2. Relation to GHCJS
3. Status & Roadmap
4. Internals
5. Conclusion

What Web backends bring to Haskell developers

1. Front-end Web programming
   - Full-stack Haskell (cf Ryan Trinkle Lambda Jam 2018 talk on Youtube)
2. Standalone applications (portable, with a GUI)
   - Node.JS engine bundled with HTML/CSS rendering engine
   - e.g. ElectronJS, NW.JS...

- Two things that current Haskell ecosystem is bad at!
  - Rated "immature" on the State of Haskell Ecosystem wiki page

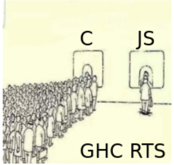
- What's really new: directly available from stock GHC
  - Not from external projects such as GHCJS or Asterius

JS vs Wasm backend: do we really need both?

- Different targets, different backend implementations, different trade-offs
- JavaScript backend's peculiarities
  - Own Runtime System written in JavaScript
  - First GHC backend to target a managed platform
  - JavaScript easy to hack and to observe (debug)
  - GHCJS already proved that JavaScript can be used in production

JS backend: a different RTS written in JavaScript

Backend or project	Runtime System (RTS)
NCG, C, and LLVM backends	C
JS backend, GHCJS	JS
Asterius	JS (not the same)
Wasm backend	C (compiled to Wasm)



- Full control of the toolchain
  - The backend doesn't rely on external tooling to convert from C to JS
  - We can have exactly the JS code we want
  - Analogous to native code generator (NCG) vs C/LLVM backends for codegen
    - More work, but more control
- Demo: show rts/js

JS backend: first to target a managed platform

- "Managed platform": own heap, heap objects, and garbage collector
- Implies changes:
  - No pointers
    - Addr# isn't represented with a number in JavaScript
  - Foreign heap objects need to be representable in Haskell codes
    - We probably need a new ManagedRef# (aka JSVal#) primitive type
  - C FFI imports should be avoided or avoidable
    - E.g. by providing fallback Haskell code (e.g. ghc-bignum's native backend)
- It will make implementing other backends easier
  - E.g. JVM and CLR (.Net)

JS backend: observability & tinkering

- Full control of the toolchain
  - No LLVM, wasi-sdk, assembler, linker. . .
  - No limitation due to external factors
- JavaScript is interpreted and (quite) readable
  - Observe and dump anything, even interactively
  - Useful for debugging
- Many JavaScript tools available
  - Profilers, debuggers...

Demo: SumInt64

1. Show sources
2. Build and run
3. Present artefacts
4. Load in Chromium perf debugger
5. Show STG
6. Primops
   - subInt64# in STG
   - primops.txt.pp: Int64SubOp
   - GHC.StgToJS.Prim: Int64SubOp
   - rts/js/arith.js: h$hs_minusInt64

Table of Contents

1 Motivation

2 Relation to GHCJS

3 Status & Roadmap

4 Internals

5 Conclusion

GHCJS: overview

- Independent project (github.com/ghcjs)
- Haskell to JavaScript compiler
- Supports full Haskell (threads, Template Haskell, finalizers. . .)
    - Compared to alternatives like Fay or Haste
- Developed since ∼2010
- Used in production
- Relies on a fork of GHC 8.x

The JS backend reused code from GHCJS

But why did we need the JS backend in the first place if we had GHCJS?

GHCJS issues and JS backend current status

GHCJS...

- ✓ ...is difficult to build (without Nix)
- ✓ ...is stuck on old GHC
- ✓ ...lacks CI
- ↗ ...lacks documentation
- ↗ ...is slow
- ↗ ...lacks maintainers
- ↘ ...produces code that is too big

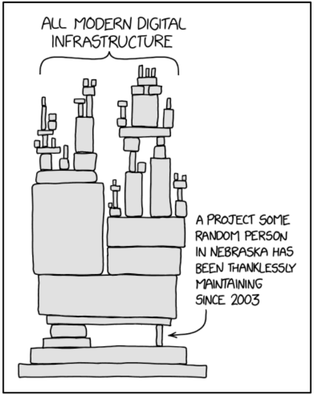
Maintainers

GHCJS' maintainers

- Victor Nazarov (2010)
- Hamish Mackenzie (2011-2013)
- Luite Stegeman (2012-2021)

JS backend's maintainers

- Jeffrey Young
- Josh Meredith
- Luite Stegeman
- Sylvain Henry
- *Add your name here*


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

https://xkcd.com/2347/

Building GHC with the JS backend

- Nearly identical to building native GHC
- You need the Emscripten C to JS toolchain
  - GHC requires a C toolchain
  - Used for configure script, platform constants (e.g. `sizeof(uint_t)`), hsc2hs, CPP...
- Don't use `--freeze1` (doesn't work for cross-compilers)

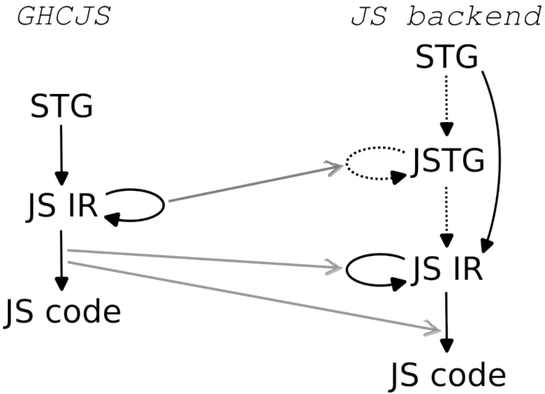
```
$ ./boot
$ emconfigure ./configure --target=javascript-unknown-ghcjs
$ ./hadrian/build --flavour=default+no_profiled_libs+native_bignum -j
```

wait ∼30 minutes, depending on your hardware

```
$ alias ghc-js=`pwd`/_build/stage1/bin/javascript-unknown-ghcjs-ghc
```

Code size regression

- We didn't fully port GHCJS' JS optimizer
    - Reason: was slow and brittle; needed redesign and rewrite



Key takeaways for GHCJS users

- The JS backend is based on GHCJS but it isn't GHCJS.
  - Expect some changes
- GHCJS support is discontinued in favor of the JS backend.
  - But it's free software, anyone can pick it up
- The JS backend isn't production-ready yet
  - Targeting GHC 9.10, cf roadmap (next topic)

Table of Contents

1. Motivation
2. Relation to GHCJS
3. Status & Roadmap
4. Internals
5. Conclusion

GHCJS upstreaming process(es)

- Before 2022: make GHCJS converge towards GHC
  - Avoid Template Haskell: e.g. replace JMacro QuasiQuoters
  - Only use boot libs: e.g. replace `lens`
  - Upgrade fork from GHC 8.6 to GHC 8.10
- Since 2022: consider GHCJS as a prototype; implement a proper JS backend reusing GHCJS' code

Roadmap: GHC 9.6

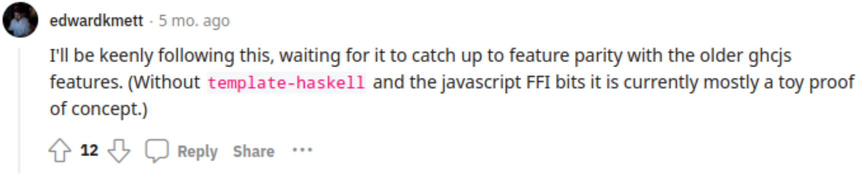
1. Generate JS code from STG

2. Building working HelloWorld program

3. Add CI, pass testsuite

Roadmap: GHC 9.8

- FFI callbacks (see GHC Users' Guide)
- Template Haskell (hopefully!)
  - MR !9779

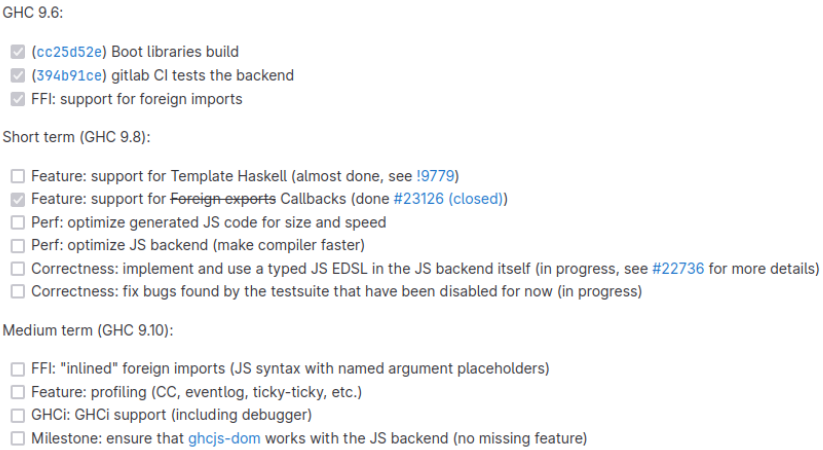


edwardkmett · 5 mo. ago

I'll be keenly following this, waiting for it to catch up to feature parity with the older ghcjs features. (Without `template-haskell` and the javascript FFI bits it is currently mostly a toy proof of concept.)

Roadmap: GHC 9.10+

Owl lift-off, via Dall-E



GHC 9.6:

- ☑ (cc25d52e) Boot libraries build
- ☑ (394b91ce) gitlab CI tests the backend
- ☑ FFI: support for foreign imports

Short term (GHC 9.8):

- □ Feature: support for Template Haskell (almost done, see !9779)
- ☑ Feature: support for ~~Foreign exports~~ Callbacks (done #23126 (closed))
- □ Perf: optimize generated JS code for size and speed
- □ Perf: optimize JS backend (make compiler faster)
- □ Correctness: implement and use a typed JS EDSL in the JS backend itself (in progress, see #22736 for more details)
- □ Correctness: fix bugs found by the testsuite that have been disabled for now (in progress)

Medium term (GHC 9.10):

- □ FFI: "inlined" foreign imports (JS syntax with named argument placeholders)
- □ Feature: profiling (CC, eventlog, ticky-ticky, etc.)
- □ GHCi: GHCi support (including debugger)
- □ Milestone: ensure that ghcjs-dom works with the JS backend (no missing feature)

https://gitlab.haskell.org/ghc/ghc/-/wikis/JavaScript-backend#roadmap

GHCJS libraries need to be updated

- GHCJS had a huge ecosystem
- Initial plan: demo with some GHCJS' GUI framework (e.g. shine)
- Not possible yet because packages need to be updated
    - ghcjs-base, ghcjs-dom, jsaddle-*...
    - Adapt from GHC 8.x to GHC 9.x
    - Need to fix almost all FFI calls (next slide)

JavaScript FFI: GHCJS inline syntax is not supported

```
-- GHCJS inline syntax: not supported!
foreign import javascript
    "$1===$2"
    js_eq :: JSString -> JSString -> Bool

-- replace with:
foreign import javascript
    "((x,y) => { return x===y; })"
    js_eq :: JSString -> JSString -> Bool
```

- Reasons
  - need to add a JS parser to GHC
  - need to coordinate with Wasm backend
  - need a GHC proposal
- In the meantime, please help updating GHCJS' libraries!

Demo: Svg

1. Build and run with native backend
2. Build and run with JS backend
    - Show how to run cabal (build.sh)
    - Run and debug
    - js-sources (+bug)
3. Load in the browser

Table of Contents

1. Motivation
2. Relation to GHCJS
3. Status & Roadmap
4. Internals
5. Conclusion

Execution model

- Lazy graph reduction: STG machine in JavaScript
  - Registers → JS global variables
  - Stack → JS array
  - Heap objects → JS values (no storage manager in the JS RTS!)
- Demo: present preamble of rts.js
  - Registers
  - Stack, sp

Heap objects

- STG objects: CONSTR, FUN, THUNK, PAP, BLACKHOLE...
  - Represented in JS as object: { `f, d1, d2, m` }
  - f: properties ("info-table") and entry function
  - d1, d2: payload
  - m: mark used for heap traversal (weak references)
- Demo: Constructors
  - `-ddisable-js-minifier`
  - Without optimization
  - Present HS, STG and JS
  - `h$log(h$mainZCMainzijust5);`
  - `dumpCAF`

Mapping of primitive types

Haskell	JavaScript
Int#, Word#, Int8#...	Number
Float#, Double#	Number
Int64#, Word64#	Two numbers (high, low)
ByteArray#...	ArrayBuffer
Array#...	Array
MVar#, ThreadID#, Weak#...	Object
Addr#	ArrayBuffer and number (offset)

- Demo: Prim
  - Show JS: `bar` returning primitive values
  - Run: 2 mallocs with same "address" result
  - Enable JS dump

Scheduler

- STG machine usually implemented with tail calls / goto
  - JS engines don't support tail calls
  - Using normal calls to implement tail calls would blow the JS call stack
- What the JS backend does:
  - Haskell functions are represented as 0-argument JS functions
    - Arguments are passed via global variables
  - They return their own continuation
  - Trampoline in the scheduler: `while(true) { c = c(); }`
- Demo:
  - show rts/js/thread.js runThreadSlice
  - "Fun" demo (without optimizations)
  - Show STG and JS for add10
  - add a bang to x in add10
  - add `h$log(c.name)`
  - add `if (c.name.match("main"))`
- https://engineering.iog.io/2023-04-21-stacks-in-the-js-backend

Template Haskell interpreter

- GHCJS pioneered the “external interpreter” idea
    - Now we can reuse the upstream one!
- Overview
    - Execute js-interp.hs in Node.js
    - Communicate via Unix pipes with it
    - Can send compiled Haskell code to load and to execute
    - First send the external interpreter code
    - Then send TH splices and their dependencies

How to implement a new GHC backend: code overview

- Declare platform: ghc-boot:GHC.Platform.ArchOS
- Fix build system: config.sub, hadrian, Cabal (js-sources), utils/deriveConstants
- Add new IR: GHC.JS.Syntax
- Add compilation pipeline: GHC.StgToJS.CodeGen
- Add FFI support: GHC.HsToCore.Foreign.JavaScript
- Fix boot libraries
  - JS: implement primitives (Posix...) using JS APIs
  - e.g. libraries/base/jsbits, js-sources, foreign import javascript
- Add linker support: GHC.StgToJS.Linker
- Add runtime system or reuse existing one: rts/js/*
- Add interpreter for Template Haskell: GHC.Runtime.Interpreter.JS and `js-interp.js`
- Hook all this into the compiler driver: GHC.Driver.{Backend...}

Table of Contents

1. Motivation
2. Relation to GHCJS
3. Status & Roadmap
4. Internals
5. Conclusion

Contribution ideas

- Build cool stuff with it and report/fix the issues you face!
- Help updating the GHCJS ecosystem and other libs (C sources...)
- Profile generated code and fix performance issues
  - E.g. replace use of BigInt from numerical primop implementation
- Help with any other item on the roadmap
  - Fix skipped tests in the testsuite (grep "js_broken")
  - Profile GHC using the JS backend and optimize it
  - (Re)implement support for profiling (cost centers...)
  - Add JS code optimizations
  - Fix tickets with JavaScript label
- Support generating TypeScript code
- Support generating JS source maps
- Implement support for delimited continuation primops
- Your ideas here...

Contact

- For assistance:
  - Open tickets on GHC's gitlab
  - ghc-devs mailing list
  - sylvain.henry@iohk.io or sylvain@haskus.fr

Work funded by IOG

Appendix

Potential IOG use cases

- Code reuse
  - Cardano blockchain network nodes written in Haskell
  - Reuse code to implement clients accessing the network
  - e.g. standalone GUI wallets, Web wallets
- Full-stack Haskell for smart contracts
  - Smart contracts fully written in Haskell
  - One part compiled to blockchain IR
  - Other part compiled to JS/Wasm to run into users' wallets (UI)