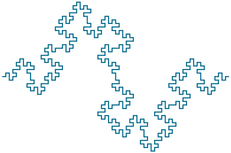
Concurrent Haskell in the browser

Luite Stegeman



August 29, 2013

GHCJS FEATURES

Compiler / runtime

- Preemptive threads
- STM
- Template Haskell
- Cabal support
- Browser and node.js, jsshell

GHCJS FEATURES

Example

```
# cat hello.hs
main = putStrLn "Hello, world"
# ghcjs -o hello hello.hs
generating native
[1 of 1] Compiling Main             ( hello.hs, hello.o )
generating JavaScript
[1 of 1] Compiling Main             ( hello.hs, hello.js_o )
Linking hello.jsexe (Main)
# node hello.jsexe/all.js
Hello, world
```

GHCJS FEATURES

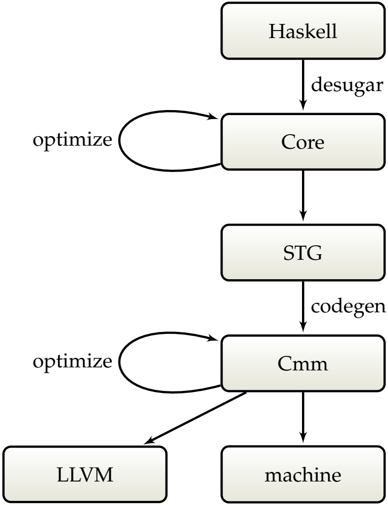
Compiler / runtime

- Preemptive threads
- STM
- Template Haskell
- Cabal support
- Browser and node.js, jsshell

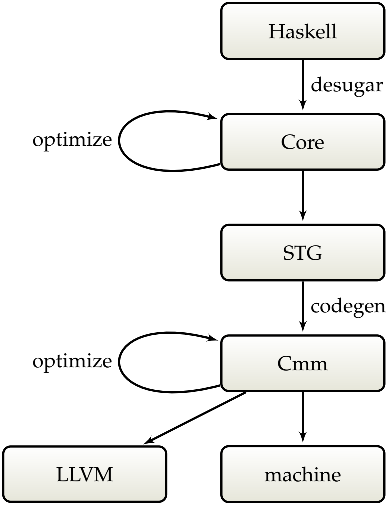
Types

- Char
- Double
- Int, Int8, Int16, Int32, Int64
- Word, Word8, Word16, Word32, Word64
- Integer
- *No single precision Float*
- *Limitations with pointers*
- No *par*

GHC PIPELINE



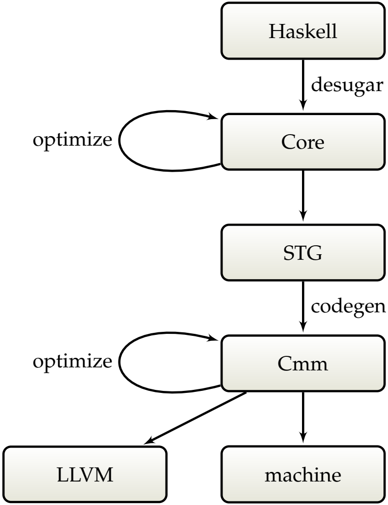
GHC PIPELINE



Haskell

- parse
- rename
- typecheck

GHC PIPELINE

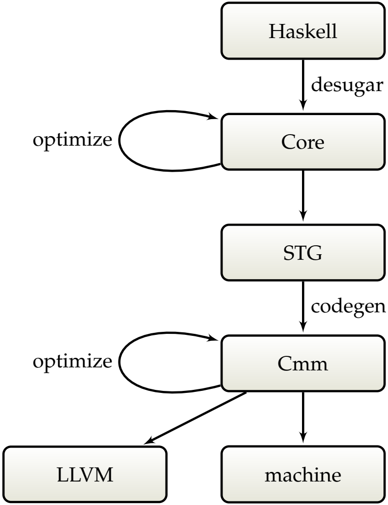


Core

based on System F with:

- algebraic data types
- let and case expressions
- type equality coercions

GHC PIPELINE



Core

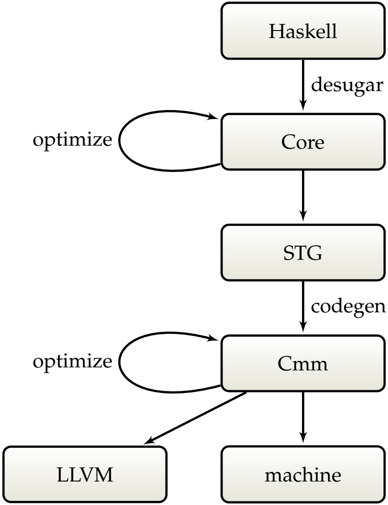
Haskell:

$$
\begin{aligned}
&\mathit{factorial} :: \mathit{Int} \to \mathit{Int}\\
&\mathit{factorial}\ 1 = 1\\
&\mathit{factorial}\ n = n * \mathit{factorial}\ (n-1)
\end{aligned}
$$

Core:

$$
\begin{aligned}
&\mathit{factorial} = \lambda n \to\\
&\quad \textbf{case}\ n\ \textbf{of}\\
&\qquad 1 \to 1\\
&\qquad n' \to n' * \mathit{factorial}\ (n-1)
\end{aligned}
$$

GHC PIPELINE



Core

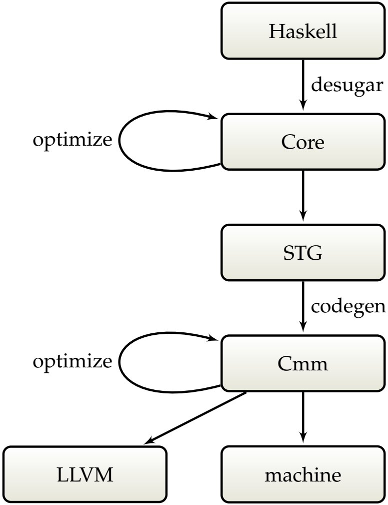
Haskell:

$$
\begin{aligned}
&\mathit{map} :: \mathit{forall}\ a\ b \circ (a \to b) \to [a] \to [b]\\
&\mathit{map}\ f\ (x : xs) = f\ x : \mathit{map}\ f\ xs\\
&\mathit{map}\ \_\ [\,] = [\,]
\end{aligned}
$$

Core:

$$
\begin{aligned}
&\mathit{map} = \lambda @a\ @b\ f\ xs \to\\
&\quad \textbf{case}\ xs\ \textbf{of}\\
&\qquad [\,] \to [\,]\ @b\\
&\qquad (y : ys) \to (:)\ @b\ (f\ x)\ (\mathit{map}\ @a\ @b\ f\ ys)
\end{aligned}
$$

GHC PIPELINE



Core

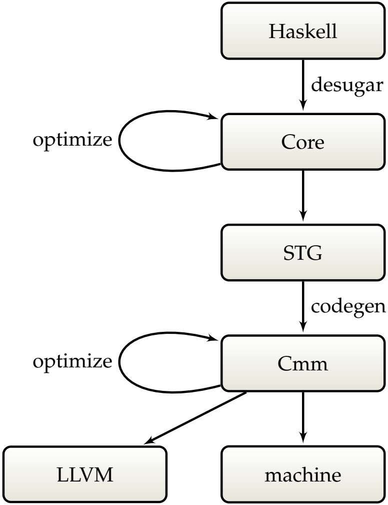
Haskell:

$$
\begin{aligned}
&\mathit{max} :: \mathit{Ord}\ a \Rightarrow a \to a \to a \\
&\mathit{max}\ x\ y \mid x \geqslant y = x \\
&\phantom{\mathit{max}\ x\ y} \mid \mathit{otherwise} = y
\end{aligned}
$$

Core:

$$
\begin{aligned}
&\mathit{max} = \lambda @a\ \$\ d\ x\ y \to \\
&\quad \mathbf{case}\ (\geqslant)\ @a\ \$\ d\ x\ y\ \mathbf{of} \\
&\qquad \mathit{False} \to y \\
&\qquad \mathit{True} \to x
\end{aligned}
$$

GHC PIPELINE



Haskell

desugar

optimize

Core

STG

codegen

optimize

Cmm

LLVM

machine

Core

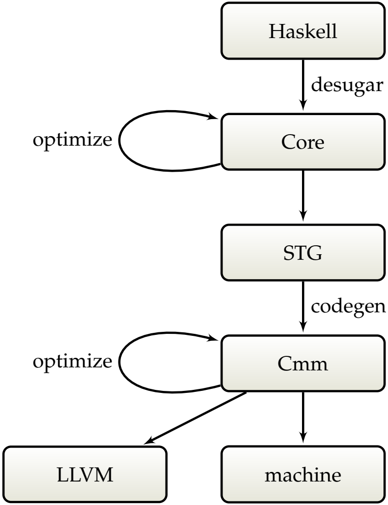
Haskell:

$$\begin{aligned}&\mathit{hyp} :: \mathit{Double} \rightarrow \mathit{Double}\\&\mathit{hyp}\ x = \mathbf{let}\ \mathit{xsq} = x * x\ \mathbf{in}\ \mathit{sqrt}\ (\mathit{xsq} + \mathit{xsq})\end{aligned}$$

Core:

$$\begin{aligned}&\mathit{hyp} = \lambda x \rightarrow\\&\quad \mathbf{let}\ \mathit{xsq} = (*)\ @\mathit{Double}\ \mathit{dictNumDouble}\ x\ x\\&\quad \mathbf{in}\ \mathit{sqrt}\ @\mathit{Double}\ \mathit{dictFloatingDouble}\ \mathit{xsq}\ \mathit{xsq}\end{aligned}$$

GHC PIPELINE



Haskell

desugar

optimize

Core

STG

codegen

optimize

Cmm

LLVM

machine

Core

Haskell:

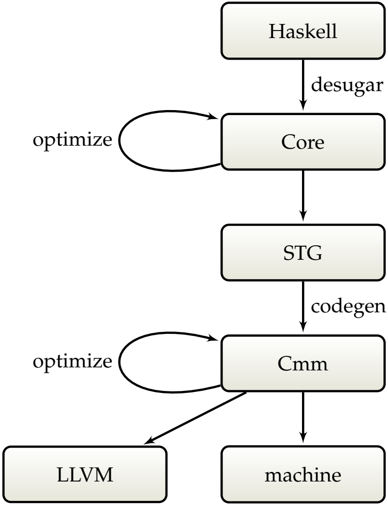
$$
\begin{aligned}
&\mathit{doNothing} :: \mathit{IO}\ ()\\
&\mathit{doNothing} = \mathit{return}\ ()
\end{aligned}
$$

Core:

$$
\begin{aligned}
&\mathit{doNothing} :: \mathit{IO}\ ()\\
&\mathit{doNothing} = \mathit{doNothing1}\ \text{‘}\mathit{cast}\text{‘}\ \mathit{someCo}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{doNothing1} :: \mathit{State}\,\#\ \mathit{RealWorld}\\
&\qquad\qquad \rightarrow (\#\mathit{State}\,\#\ \mathit{RealWorld}, ()\#)\\
&\mathit{doNothing1} = \lambda s \rightarrow (\#s, ()\#)
\end{aligned}
$$

GHC PIPELINE

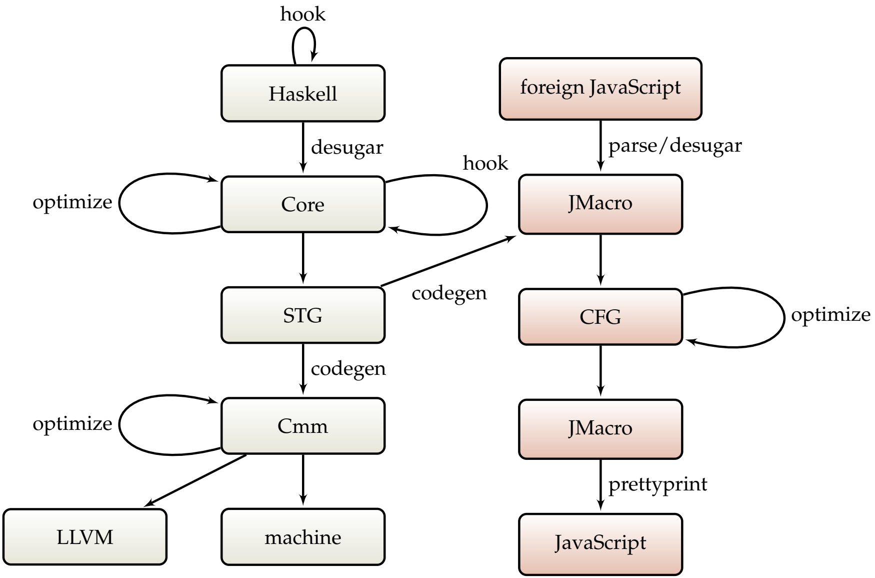


STG

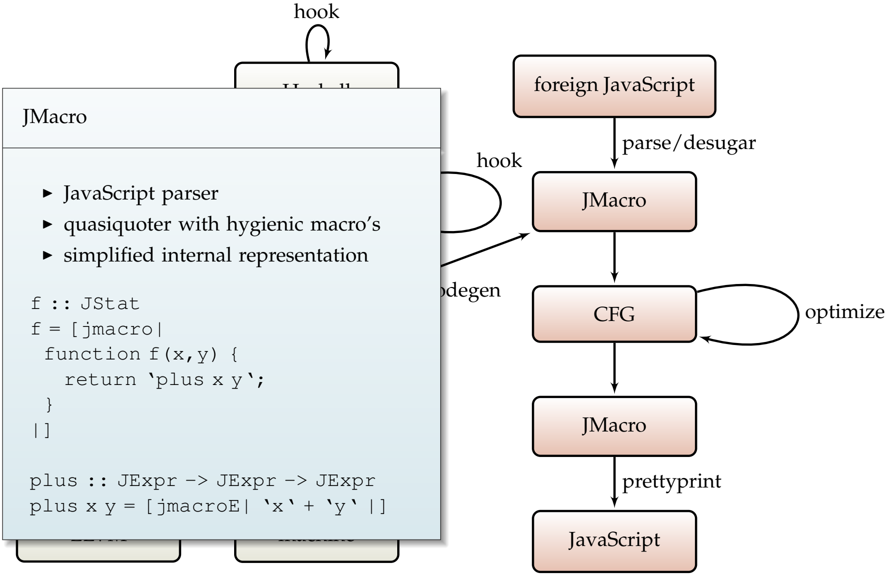
Spineless Tagless G-machine

- A-normal form
- Primops and data constructors saturated
- free variable annotations

GHC PIPELINE



GHC PIPELINE



JMacro

- JavaScript parser
- quasiquoter with hygienic macro's
- simplified internal representation

```
f :: JStat
f = [jmacro|
 function f(x,y) {
   return `plus x y`;
 }
|]

plus :: JExpr -> JExpr -> JExpr
plus x y = [jmacroE| `x` + `y` |]
```

GENERATING CODE

- Primops
- Let
- Case
- Foreign imports

GENERATING CODE

Primops

```
prim DoubleGtOp [r] [x, y] =
   Inline [jmacro | ‘r‘ = (‘x‘ > ‘y‘) ? 1 : 0 |]

prim ReadByteArrayOp_Int32 [r] [a, i] =
   Inline [jmacro | ‘r‘ = ‘a‘ ∘ i3 [‘i‘]; |]

prim BSwap16Op [r] [x] =
   Inline [jmacro | ‘r‘ = ((‘x‘ & 0 xFF) << 8) | ((‘x‘ & 0 xFF00) ≫ 8); |]

prim TakeMVarOp [r] [m] =
   OutOfLine [jmacro | return h $ takeMVar (‘m‘); |]
```

GE

Foreign imports

Ccall compatibility:

```
foreign import ccall "sin" c_sin :: Double → Double
```

Extended syntax:

```
foreign import javascript "Math.sin($1)"
  js_sin :: Double → Double
foreign import javascript interruptible
  "jQuery.ajax($1,$2).always(function(d,ts,xhr) {"
           "if(typeof(d) === 'string') {"
           "  $c({ data: d, status: xhr.status });"
           "} else {"
           "  $c({ data: null, status: d.status });"
           "}"
       ");"
  jq_ajax :: JSString
          → JSRef ajaxSettings
          → IO (JSRef ajaxResult)
```

GENERATING CODE

- Primops
- Let
- Case
- Foreign im

Foreign imports

- Bool passed as true/false
- JSRef type

Safety:

- *safe*: JavaScript exceptions converted to Haskell
- *unsafe*: JavaScript exception kills thread
- *interruptible*: Async FFI (JS calling convention only)

GENERATING CODE

- Primops
- Let
- Case
- Foreign imports

$$
\begin{aligned}
&\mathit{length} :: [a] \rightarrow \mathit{Int}\\
&\mathit{length}\ [\,] = 0\\
&\mathit{length}\ (\_ : \mathit{xs}) = 1 + \mathit{length}\ \mathit{xs}
\end{aligned}
$$

```
function length(a) {
 var a_ = force(a);
 if(a_.constructor === 1) {
  return con(int, 0);
 } else {
  var xs = a_.field1;
  var l  = ap1(length, xs);
  return force(ap2(plusInt,
    con(int, 1), l));
 }
}
```

```
function force(thunk) {
 if(!thunk.f) return thunk.r;
 var f = thunk.f;
 thunk.f = null;
 thunk.r = f();
 return thunk.r;
}
function ap2(force,a,b) {
 return { f: function() {
              return force(fun)(a)(b);
             }
        , r: null;
        }
}
```

GENER

$$\begin{aligned}
&\mathit{length} :: [a] \rightarrow \mathit{Int}\\
&\mathit{length}\ [\,] = 0\\
&\mathit{length}\ (\_ : \mathit{xs}) = 1 + \mathit{length}\ \mathit{xs}
\end{aligned}$$

```
function length(a) {
 force(a, function(a_) {
  if(a_.constructor === 1) {
   return con(int, 0);
  } else {
   var xs = a_.field1;
   var l  = ap1(length, xs);
   return force(ap2(plusInt,
     con(int, 1), l)));
  }
 })
}
```

- I
- I
- C
- I

Gener

- I
- I
- C
- I

```
function length() {
  stack.push(length1);
  return force(arg1);
}
function length1() {
 stack.pop();
 if(arg1.constructor === 1) {
  arg1 = con(int, 0);
  return stack[stack.length-1];
 } else {
  return (force(ap2(plusInt,
   con(int, 1), l)));
 }
}

function mainloop(c) {
 while(c) c = c();
}
```

Mapping Haskell types to JS

Haskell	JavaScript	
Bool	*boolean*	
Char#, Char	*number*	
IntPrim, Int	*number*	
Word#, Word	*number*	stored as signed
Int64#	*number* × *number*	stored as signed
Word64#	*number* × *number*	stored as signed
ByteArray#	*typed array*	
Addr#	*typed array* × *number*	data plus offset
other	*object*	
Integer	*JSBN*	sign field unused

OPTIMIZING

- How does the generated code look?

```
function f() {
 var a = h$r1.d1;
 var b = h$r1.d2;
 var c = b.d1;
 var d = b.d2;
 var e = b.d3;
 var f = b.d4;
 var g = b.d5;
 var h = b.d6;
 var i = b.d7;
 h$bh();
 var j = ((i === g) ? 1 : 0);
 var k = (j ? true : false);
 if(k) {
  return h$e(h);
 } else {
  var l = h$c7(buffer_con_e,
   a, c, d, e, f, i, g);
  h$r1 = l;
  return h$stack[h$sp];
 };
};
```

look?

OPTIMIZING

- How does the generated code look?
  - many redundant assignments
  - awkward primop types

OPTIMIZING

- How does the generated code look?
  - many redundant assignments
  - awkward primop types
- Making it better: Dataflow analysis
  - constant propagation
  - liveness
  - per function, using RTS knowledge

OPTIMIZING

- How does the generated code look?
  - many redundant assignments
  - awkward primop types
- Making it better: Dataflow analysis
  - constant propagation
  - liveness
  - per function, using RTS knowledge
- The CFG type
  - keep JMacro AST structure
  - all break/continue statement targets resolved
  - node annotations for performance

```
function f() {
 var a = h$r1.d1;
 var b = h$r1.d2;
 var c = b.d1;
 var d = b.d2;
 var e = b.d3;
 var f = b.d4;
 var g = b.d5;
 var h = b.d6;
 var i = b.d7;
 h$bh();
 var j = ((i === g) ? 1 : 0);
 var k = (j ? true : false);
 if(k) {
  return h$e(h);
 } else {
  var l = h$c7(buffer_con_e,
   a, c, d, e, f, i, g);
  h$r1 = l;
  return h$stack[h$sp];
 };
};
```

Optimized

```
function f() {
 var a = h$r1.d1;
 var b = h$r1.d2;
 var g = b.d5;
 var i = b.d7;
 h$bh();
 if((i === g)) {
  return h$e(b.d6);
 } else {
  h$r1 = h$c7(buffer_con_e, a,
   b.d1, b.d2, b.d3, b.d4, i, g);
  return h$stack[h$sp];
 };
```

nt targets resolved

mance

Linking

- Start with set of root functions, callable from JavaScript
- Follow function-level dependencies
- Combine result, compact metadata
- Collect foreign library dependencies
- Generated names start with h$ or h$$ to make optional renaming easy
- Output:

all.js	bundle of everything (runnable with node.js)
out.js	the compiled Haskell code
rts.js	generated RTS
lib.js	foreign libraries
lib.js.files	files in *lib.js*
lib1.js	foreign libraries (to be included after RTS)
lib1.js.files	files in *lib1.js*

LINKING

- Start with set of root functions, callable from JavaScript
- Follow function-level dependencies
- Combine result, compact metadata
- Collect foreign library dependencies
- to make optional
- nnable with node.js)
  de

From HTML

```
h$main(h$mainZCMainzimain);
h$run(h$mainZCMainzimain);
h$runSync(h$mainZCMainzimain);
```

lib.js	foreign libraries
lib.js.files	files in *lib.js*
lib1.js	foreign libraries (to be included after RTS)
lib1.js.files	files in *lib1.js*

HACK ON GHCJS!

You need:

- GHC HEAD with GHCJS patch
- Cabal with GHCJS patch
- Lots of packages updated to work with GHC HEAD

Vagrant 1.2 virtual machine:

- prebuilt: 450MB archive with binaries
- regular: everything from source, 90 minutes to build

TASKS

- Support JavaScript library dependencies with Cabal
- Implement foreign code for packages
- Bindings for JavaScript libraries
- Incremental linking
- On-demand code loading
- Extend the FFI
- Port non-concurrent backend to JMacro