Web browser programming with UHC's JavaScript backend

Atze Dijkstra, Jurriën Stutterheim, Alessandro Vermeulen, and Doaitse Swierstra

Haskell Implementors Workshop, Sep 14, 2012

“The JavaScript problem”

- JavaScript has several shortcomings
  - Dynamic, weak typing
  - Verbose syntax
  - Peculiar equality and scoping rules
- JavaScript is the *only* cross-browser language
  - Or use alternatives: plugins, Java applet, modify browser...

(`http://www.haskell.org/haskellwiki/The_JavaScript_Problem`)

UHC JavaScript backend

Use JavaScript as a high-level "machine" language for targeting Haskell to

- And exploit freedom available in FFI entity strings

Alternative approaches

- Based on GHC: Haste, GHCJS
- (Javascript compilers for Haskell subsets: haskellinjavascript)
- (Haskell functionality merged into Javascript: Functional Javascript)
- (Already previously done: YHC)

(`http://www.haskell.org/haskellwiki/The_JavaScript_Problem`)

Other (potential) benefits

- Libraries can be used on both client and server
  - Allows solutions used in Clean system (iTasks)
- Eliminate AJAX calls, improving responsiveness
- Use QuickCheck for indirectly testing JavaScript code
- ...

This talk

Content

- Implementation machinery
- Interaction with Javascript
  - Foreign function interface
  - Embedding in Html
  - Platform specific library
  - Using objects
- JCU application
- Lessons

Implementation machinery

Represent laziness by wrapper objects around Javascript functions + explicit evaluation

- Functions: `new _F_(function (..)  {..})`
- Function application: `new _A_(new _F_(..), [..])`
- Evaluation: `_e_(..)`

Plain Javascript values are recognized by the evaluator

Implementation machinery

Example

- Haskell

$$add3\ x\ y\ z = x + y + z$$

- JavaScript: function

```
var add3 = new _F_
  (function (x, y, z) {return x + y + z;});
```

- JavaScript: application

```
var app345 = new _A_(add3, [3, 4, 5]);
```

- JavaScript: evaluation

```
var answer = _e_(app345);
```

Interacting with JavaScript

- Useful programs need to interact with plain JavaScript (DOM, libraries)
- Impedance mismatch: strict, imperative, OO vs. lazy, purely functional
- Use the Foreign Function Interface (FFI) with JavaScript calling convention
- Foreign Expression Language (FEL) to partly overcome impedance mismatch

Importing a JavaScript function

JavaScript

```
someStr.subString(start, length);
```

Haskell

foreign import *js* `"%1.subString(%2, %3)"`
  $subString :: JSString \to Int \to Int \to JSString$

$JSString$: Haskell type for a JavaScript string.

`dynamic` and `wrapper` imports work as expected.

Exporting a Haskell function

Haskell

$$mySum :: Int \to Int \to Int$$
$$mySum\ x\ y = x + y$$
$$\textbf{foreign export}\ js\ \texttt{"mySum"}\ mySum :: Int \to Int \to Int$$

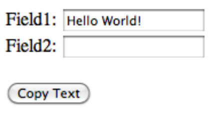
JavaScript

```
var mySum = function(x, y) {
  return _e_(new _A_(haskMySum, [x, y])); }
```

Javascript in a browser

Example: Copy text between fields

- Browser:



- Usual Html:

```
<!DOCTYPE html> <html> <head> <script>
function copyText()
{ document.getElementById("field2").value =
    document.getElementById("field1").value; }
</script> </head> <body>
Field1: <input type="text" id="field1" value="Hello World!"/
Field2: <input type="text" id="field2" />
<br /><br />
<button onclick="copyText()">Copy Text</button>
</body> </html>
```

Javascript in a browser

In Haskell

```
module HtmlDomUse where

import Language.UHC.JS.Prelude
import Language.UHC.JS ∘ W3C.HTML5

copyText :: IO ()
copyText = do
  d  ← document
  n1 ← documentGetElementById d (toJS "field1")
  n2 ← documentGetElementById d (toJS "field2")
  elementSetAttribute n2 "value"
    (fromJS (elementValue n1))

foreign export js "copyText" copyText :: IO ()

main = return ()
```

Javascript in a browser

Html loads generated code

```
<!DOCTYPE html> <html>
<script type="text/javascript" src="HtmlDomUse.js"></script>
<head> </head> <body>
...
</body> </html>
```

JavaScript objects

The problem

- Existing JavaScript APIs expect and return objects
- How do we represent, create, query, and manipulate JavaScript objects in a purely functional language?

Representing objects

- JavaScript objects are represented as an opaque pointer type $JSPtr\ a$
- This type has no constructors, so objects can only be obtained via the FFI

Creating, querying, and manipulating objects

- Use FFI accessible JavaScript functions that wrap around JavaScript's object syntax as primitive functions
- Result: object interaction with a functional flavour
- Imported and exposed via a UHC specific JavaScript library

Primitives: creating JavaScript objects

Instantiate an object of a given constructor, creating the constructor if needed:

$$mkObj :: JSString \rightarrow IO\ (JSPtr\ a)$$

Instantiate an anonymous object ({} in JavaScript)

$$mkAnonObj :: IO\ (JSPtr\ a)$$

Primitives: querying and modifying objects

$$
\begin{aligned}
getAttr &:: JSString \qquad\qquad\qquad \rightarrow JSPtr\ b \rightarrow IO\ a \\
getAttr &:: JSString \rightarrow a \qquad\qquad \rightarrow JSPtr\ b \rightarrow IO\ (JSPtr\ b) \\
modAttr &:: JSString \rightarrow (a \rightarrow b) \rightarrow JSPtr\ c \rightarrow IO\ (JSPtr\ c)
\end{aligned}
$$

- Similar primitives are available for prototype attributes
- Extensive use of IO due to JavaScript's mutable nature
- Loss and gain of type-safety
  - Low level primitives are polymorphic
  - Restricting types delegated to caller of primitives
  - $JSPtr\ a$ not a phantom type, type may be freely chosen but is supposed (!) to stand for actual Javascript object (proto)type

Pure variants

Pure operations can be simulated by cloning an object and modifying the clone:

$$primClone :: JSPtr\ a \to JSPtr\ a$$

Which allows pure (albeit inefficient) mutator functions:

$$\begin{aligned} pureSetAttr &:: JSString \to a \to JSPtr\ b \to JSPtr\ b \\ pureModAttr &:: JSString \to (a \to b) \to JSPtr\ c \to JSPtr\ c \end{aligned}$$

Creating objects

Create empty object, then set attributes

```
main :: IO ()
main = do
  b ← mkObj "Book"
  setAttr "author" "Lipovaca" b
  setAttr "title" "LYAH" b
  setAttr "pages" 400 b
  setAttr ...
  ...
```

Somewhat laborious

JavaScript objects and Haskell datatypes

Haskell constructors are very similar to JavaScript objects

$$
\begin{aligned}
&book\\
&\quad = Book\\
&\quad \{\ author = toJSString\ \texttt{"Lipovaca"}\\
&\quad ,\ title \quad = toJSString\ \texttt{"LYAH"}\\
&\quad ,\ pages \ = 400\}
\end{aligned}
$$

```
book
=
{ author :  "Lipovaca"
, title  :  "LYAH"
, pages  :  400 }
```

Automatic conversion

Special object wrapper import

$$\textbf{foreign import}\ js\ \texttt{"\{\}"}$$
$$\quad toObj :: a \rightarrow IO\ (JSPtr\ b)$$

Knows constructor implementation, converts (at runtime) from datatypes to JavaScript objects

$$main = \textbf{do}$$
$$\quad \textbf{let}\ b' = book\ \{pages = pages\ book + 1\}$$
$$\quad b \leftarrow toObj\ b'$$
$$\quad p \leftarrow getAttr\ \texttt{"pages"}\ b$$
$$\quad print\ p \quad \text{-- Prints 401}$$

Use case: JCU App

Web application for teaching about proofs and unification by dragging and dropping Prolog rules on a Prolog query

- Heavy use of JavaScript
- Ported the entire front-end application to Haskell
- Retained all functionality
- Interface with jQuery for DOM manipulation, drag & drop

Online: `http://jcu.chrisdone.com/`
(Courtesy Chris Done)

Use case: JCU App

- Eliminated several AJAX request by using Haskell libraries client-side
- Performance reasonable to good on WebKit-based browsers, slow to reasonable on others
- Excessive Prolog backtracking extremely slow compared to native Haskell
- Risk of infinite recursion hanging application, due to current lack of threading

Lessons

Or: hurdles and challenges

- Execution platform variation
  - Artefact location
  - (In)valid libraries and (regression) tests
- Advanced language features
- ...

Lesson

Execution platform variation: artefact location

- UHC caters for multiple (virtual) machine + platform combinations
  - Artefacts (.hi, .o, .etc) end up in different locations
  - Different paths through compiler
- But...
  - Managing artefacts usually is done by a build system
- Cabal
  - Has no knowledge of target + platform, so no UHC compilation for Javascript via cabal
  - Possible solution: cater for ‘way’, distinguishing non-combinable (linkable) artefacts

And then there is Android, iOS, Java/JVM, ...

Lesson

Execution platform variation: (in)valid libraries and tests

- Different platform
  - Different available functionality
  - Different sets of available libraries
  - Library may partially work (e.g. base)
  - Different sets of valid regression tests
- UHC (ad-hoc) uses `{-# EXCLUDE_IF_TARGET js #-}`
  - Similar mechanism for regression test exclusion
- Possible solution: platform info can/must be specified by programmer
  - In: Haskell source, build (cabal) file, test, ...
  - Has meaning for various tools (compiler, build system, ...)

Lesson

Paradoxically, succes of advanced features

- Many 'desirable' libraries use non-standard features
  - Type families, template haskell, ...
  - Even base library: uses/defines extensible exceptions, which use existentials packing class instances with data
- Difficult, if not impossible to keep up, yet there may be value in pluriformity/variety
- Possible solution:
  - Define base library against API for compiler provided/required minimal functionality, i.e. split base into per compiler base and compiler independent base
  - Limit base libraries to comply to a standard or fixed (minimal) set of extensions

To do

UHC specific (future work)

- Optimizations, language features, ...

Javascript specific

- Deployment: linking/loading, minimizing code size, obfuscation

Combination

- Portable GUI library/tools
  - Not just wrapping around platform specific one, like e.g. wxHaskell
- Threading, Web Workers, AJAX style client/browser communication

Conclusion

The good news

- It works!

The bad news

- It needs work!

More info...

- `https://github.com/UU-ComputerScience`
- `http://uu-computerscience.github.com/uhc-js/`