README

Sam Auciello

12 February 2013

Contents

1 HOT COCOA LISP

Hot Cocoa Lisp is a dialect of Lisp designed to be compiled into Javascript. It has been said that Javascript is Scheme in C's clothes. Hot Cocoa Lisp is an attempt to put it back in Schemes clothes.

1.1 High Level Design Goals

- Provide a clean, intuitive interface to existing Javascript libraries/frameworks
- Use the minimal syntax of Lisp

- Retain the good features of Javascript (first-class functions with closures, Dynamic objects with prototypes, Object literals).
- Fix the problems of Javascript. (Global variables, type coersion, etc.)
- Provide some of the introspective power of Lisp (quoting, macros etc.)

1.2 Syntax

The syntax of Hot Cocoa Lisp is the same as that of Scheme with four exceptions:

- As in clojure the *unquote* symbol is ~ instead of /,/.
- The square bracket notation for Array literals from JavaScript may be used but the commas are optional.

```
;; HCL
[1 2 3]
// JavaScript
[1, 2, 3]
```

  It is translated by the parser to the equivalent *(list 1 2 3)*.

- The curly bracket notation for object literals from JavaScript may be used but the commas and colons are optional.

```
;; HCL
{a 1 b 2}
// JavaScript
{a: 1, b: 2}
```

  It is translated by the parser to the equivalent *(object a 1 b 2)*.

- As in Javascript . can be used as an object access operator. It works precisely as it does in JavaScript.

```
;; HCL
foo.bar
// JavaScript
foo.bar
```

It is translated by the parser to the equivalent *(. foo bar)*.

It is also possible to use a lisp expression for the object being accessed:

```
;; HCL
(some expression returning an object).keyname
// JavaScript
some(expression, returning, an, object).keyname
```

Hot Cocoa Lisp can syntactically supports most of the literals from Javascript:

- *undefined*, *null*, *true*, and *false* work exactly as they do in JavaScript.
- Numbers work just as they do in JavaScript.
- Strings literals must use double quotes but otherwise function as they do in JavaScript.
- Functions, objects, arrays, and regular expressions are defined by the built-in functions named *#*, *object*, *list*, and *re* respectively.

1.3 Semantics

The basic logic for converting a lisp expression into compiled JavaScript is quite simple. If the first element of the expression is a built-in function the compiler calls on internal logic to compile the expression, otherwise each element is recursively compiled and then output in the form *elem1(elem2, elem3, etc)*. Most of the remaining semantic logic takes the form of built-in function definitions.

1.3.1 nop

Takes no arguments and returns undefined.

```
;; HCL
(for ((var i 0) (< i 10) (++ i))
    (nop))
// JavaScript
for (var i = 0; i < 10; i++) { }
```

1.3.2 .

Takes two or more arguments and does chained object access with keys that were known at program writing time.

```
;; HCL
(. foo bar baz)
;; OR
foo.bar.baz
// JavaScript
foo.bar.baz
```

1.3.3 get

Takes exactly two arguments and does object/list access using a key that might not be known until run time.

```
;; HCL
(get list 10)
// JavaScript
list\footnote{FOOTNOTE DEFINITION NOT FOUND: 10 }
```

Synonyms: *nth*

1.3.4 list

Create and return a new JavaScript array containing the given arguments.

```
;; HCL
(list 1 2 3)
;; OR
[1 2 3]
// JavaScript
[1, 2, 3]
```

Synonyms: *array*

1.3.5 object

Takes an even number of arguments which it interprets pairwise as key/value pairs for a new JavaScript object which it creates and returns.

```
;; HCL
(object name "Sam" age 23)
;; OR
{ name "Sam" age 23 }
// JavaScript
{name: "Sam", age: 23}
```

1.3.6 inherit

Takes a single argument and returns a new empty object that inherits prototypically fromt the argument.

```
;; HCL
(def x { a 1 })
(def y (inherit x))
y.a ; 1
// JavaScript
var x = {a: 1};
var y = Object.create(x);
y.a // 1
```

1.3.7 if

Takes three arguments and returns the second if the first is true and the third if the first is false.

```
;; HCL
(if (= x 1) "one" "not one")
// JavaScript
((x === 1) ? "one" : "not one")
```

1.3.8 when

Takes two or more arguments and executes the expressions following the first argument if the first argument is true

```
;; HCL
(when (= x 1) (console.log "x is one"))
// JavaScript
((x === 1) && (function() { console.log("x is one"); })())
```

1.3.9 cond