Cobwebs of Callbacks in JavaScript

JavaScript is single-threaded, so responsive interactive applications are written in event-driven style using callbacks.

However, standard callback composition is convoluted, tedious, and not modular.

Example: Drag-and-Drop

Plumbing is convoluted: each handler disables itself and sets up the next callbacks:

```
function setup(event) {
  var target = event.currentTarget;
  /* setup drag-and-drop */
  target.addEventListener("mousedown", setup, false);
  target.addEventListener("mousemove", drag,
                        setup, false);
  target.addEventListener("mouseup", drop, false);
  target.addEventListener("mouseout", cancel, false);
}
```

Modification is tedious and error-prone:
1. repeating drag-and-drop requires modifying drag and cancel;
2. adding a new state rotate requires modifying drag and drop.

Reusing parts of drag-and-drop is practically impossible due to hard-coded next states.

Arrows Point the Way…

…from Haskell…

```
instance Arrow (→→) where
  arr f = f
  (f . g) x = f (g x)
add x = x + 1
add2 = add . add1
result = add2 (−− returns 2 −−)
```

…to JavaScript (quite nicely too!)

```
function.prototype.A = function() { /* arr */
  return this;
}
function.prototype.next = function(g) { /* see */
  var f = this; g = g(A); /* ensure g is a function */
  return function(x) { return g(f(x)); }
}
function.add1(x) { return x + 1; }
function.add2 = add1.next(add1);
function.result = add2(1); /* returns 3 */
```

Arrows: Arrows for JavaScript

AsyncA prototype arrow wraps CPS functions

```
function AsyncA(cps) {
  this.cps = cps; /* cps :: (x, k) → (y, p) */
  AsyncA.prototype.AsyncA = function() { return this; }
}
AsyncA.prototype.next = function(g) { /* see */
  var f = this; g = g(AsyncA);
  return new AsyncA(cps(x, k) {
    f(cps(x, function(y) { g(cps(y, k)); });
  });
}
AsyncA.prototype.run = function(x) {
  this.cps(x, function(y) { instance});
  return p;
}
function.prototype.AsyncA = function() { /* arr */
  var f = this;
  return new AsyncA(cps(x, k) { k(f(x)); });
}
```

Prototype constructor

```
AsyncA(cps) {
  this.cps = cps;
  /* cps :: (x, k) → (y, p) */
}
```

Type identity

```
AsyncA.prototype.AsyncA = function() { return this; }
```

CPS composition

```
"typecheck" and automatically lifts functions to arrows.
```

CPS invocation

```
AsyncA.prototype.run = function(x) {
  this.cps(x, function(y) { return p; });
}
```

CPS arrow lifting

```
function AsyncA(cps) {
  this.cps = cps;
  /* cps :: (x, k) → (y, p) */
  AsyncA.prototype.AsyncA = function() { return this; }
}
AsyncA.prototype.next = function(g) { /* see */
  var f = this; g = g(AsyncA);
  return new AsyncA(cps(x, k) {
    f(cps(x, function(y) { g(cps(y, k)); });
  });
}
```

Drag-and-Drop with Arrows

Handlers written as normal functions like before, but contain no plumbing.

Drag-and-drop can be composed modularly using combinators such as next, or, and repeat:

```
function setupA(event) { /* setup drag-and-drop */
  return event.currentTarget;
}
```

```
function dragAndDropA =
  (EventA("mousedown").next(dragA).next(Repeat)
   .or(EventA("mouseup").next(dropA).next(Done))
   .or(EventA("mouseout").next(cancelA));)
```

or: allows only the first triggered arrow to complete repeat:
```
loop while Repeat(x); stop when When(x)
```

Arrow composition looks just like a state diagram, making it easy to understand.

Parts of drag-and-drop can be easily reused:

```
function jigsawA =
  (nextPieceA
   .next( (dragOrDropA.next(repeatIfWrongPlaceA)).repeat() )
   .repeat);
```

Drag-and-Drop with Arrows

```
function setupA(event) { /* setup drag-and-drop */
  return event.currentTarget;
}
```

```
function dragAndDropA =
  (EventA("mousedown").next(dragA).next(Repeat)
   .or(EventA("mouseup").next(dropA).next(Done))
   .or(EventA("mouseout").next(cancelA));)
```

or: allows only the first triggered arrow to complete repeat:
```
loop while Repeat(x); stop when When(x)
```

Arrow composition looks just like a state diagram, making it easy to understand.

Parts of drag-and-drop can be easily reused:

```
function jigsawA =
  (nextPieceA
   .next( (dragOrDropA.next(repeatIfWrongPlaceA)).repeat() )
   .repeat);
```