Composition Functions

ES2015 introduced Generators

- Push/Pull control flow
- Powerful general-purpose feature
- Flexible, but specifically intended for…
  - asynchrony
  - lazy computation

ES2015 + task.js

```
function getStockPrice(name) {
    return spawn(function*() {
        var symbol = yield getStockSymbol(name);
        var price = yield getStockPrice(symbol);
        return price;
    });
};
```

ES2016: Async/Await Proposal

```
async function getStockPrice(name) {
    var symbol = await getStockSymbol(name);
    var price = await getStockPrice(symbol);
    return price;
};
```

Async/Await

- Sequences Promises using generator fn
- **await** hides generator mechanism
- Addresses very common use-case in JS

Async/Await Concerns

- Syntactic Space allocated only to Promises
- Sequencing is general operation that could also be applied to other async values
    - Task (cancellable async operation)
    - Observable

Can we accomplish the same thing with simpler primitives?

`await is then`

```
then = (M a -> (a -> b | M b) -> M b)
bind = (M a -> (a -> M b)     -> M b)
```

The await keyword sequences scalar Monads.

Introducing Composition Functions (CFs)

ES2016: CFs

```
Promise function getStockPrice(name) {
    var symbol = await getStockSymbol(name);
    var price = await getStockPrice(symbol);
    return price;
};
```

async await and CF

```
async function getStockPrice(name) {
      var symbol = await getStockSymbol(name);
      var price = await getStockPrice(symbol);
      return price;
};
```

```
Promise function getStockPrice(name) {
      var symbol = await getStockSymbol(name);
      var price = await getStockPrice(symbol);
      return price;
};
```

async await or CF

```
async function getStockPrice(name) {
      var symbol = await getStockSymbol(name);
      var price = await getStockPrice(symbol);
      return price;
};
```

```
var async = Promise;

async function getStockPrice(name) {
      var symbol = await getStockSymbol(name);
      var price = await getStockPrice(symbol);
      return price;
};
```

ES2016: Composition Functions

```
Promise function getStockPrice(name) {
    var symbol = await getStockSymbol(name);
    var price = await getStockPrice(symbol);
    return price;
};
```

ES2016: Composition Functions

```
function getStockPrice(name) {
    return Promise[Symbol.compose](function*() {
        var symbol = yield getStockSymbol(name);
        var price = yield getStockPrice(symbol);
        return price;
    });
}
```

Composition Functions

- Use generators for scalar monadic composition
- Extensible to new types in user-land
- Semantics of await dictated by composition function, <u>not</u> language

Prior Art

- (Weak) Similarity to F# Computation Expressions
- General-purpose Monadic syntax in other languages

Proof of Concept: Task Composition

getStockPrice Task function

```
Task function getStockPrice(name) {
    var symbol = await getStockSymbol(name);
    var price = await getStockPrice(symbol);
    return price;
}

var subscription =
    getStockPrice(‘Johnson and Johnson’).
        get(value => console.log(value),
            error => console.error(error));

// cancel task
subscription.dispose();
```

Grammer

```
CompositionFunctionDeclaration :
    Expression [no LineTerminator here] function BindingIdentifier ( FormalParameters ) { FunctionBody }

CompositionFunctionExpression :
    Expression [no LineTerminator here] function BindingIdentifier? ( FormalParameters ) { FunctionBody }

CompositionMethod :
    Expression PropertyName (StrictFormalParameters)  { FunctionBody }

CompositionArrowFunction :
    Expression [no LineTerminator here] ArrowParameters [no LineTerminator here] => ConciseBody
```

Questions

- whither await*?
- alternate syntax to reflect more abstract operation
- allow limiting to arrow expressions?

Alternate Syntax

```
Promise function getStockPrice(name) {
    var symbol = on getStockSymbol(name);
    var price = on getStockPrice(symbol);
    return price;
};
```

Priorities

- Reconcile with async/await
- Stage?