Strictness Scoping

Andreas Rossberg @ TC39, 2015/07/29

Context

- VMs need to do parsing & static checks in single pass
- ...without building an AST (lazy compilation)
- Backtracking is not an option (at least not for V8)

Easy in ES5

```
‘use sloppy’;
function f(x, x) { ‘use strict’ }
```

More difficult in ES6

```
‘use sloppy’;
function f(g = (o) => { with (o) {} }) { ‘use strict’ }
```

More difficult in ES6

```
‘use sloppy’;
function f(g = function(h) {
    { function h() {} }  return h
}) {
  ‘use strict’
}
```

Much more difficult in ES6

- The directive can affect **arbitrary** code
- Nested arbitrarily deep
- Would need to defer any sort of mode-specific decisions in the parser for code that occurs in parameters
- With arrow functions, we do not even know (in time) whether we are inside parameters

Even worse with arrows

```
‘use sloppy’;
let f = (g = () => { /* ? */ ... }, ...) => { ‘use strict’ }
```

Categories of mode specific logic

1. Mode-specific errors (e.g., ‘with’, ‘delete’, for-in, octals, ‘let’, variable name validity, parameter conflicts)
   => Easy to defer, at least in principle, but may have measurable cost.
2. Special handling of eval (scoping, variable modes)
   => Not an issue, cannot depend on local directive in same parse
3. Actual divergence in parsing/scoping (e.g., Annex B function scoping, parsing of \`yield\`)
   => Hairy, affect downstream decisions, would have to transitively defer.

Analysis

- It’s a pain to implement
- It costs performance (parsing is a bottleneck!)
- ...for many programs *not* using the feature (e.g. ES5)
- Paints us into a corner (will affect any mode-related design decision we ever going to make in the future)
- And all that for an edge case

Suggestion

Make it an error to have a ‘use strict’ directive in a function with a non-simple parameter list.

Possible Variations

- Only an error when outer mode is not strict already (refactoring trap?)
- Only when parameter list contains expressions (too complicated a rule?)