Ecma/TC39/2010/035

Overview

Generator expressions were introduced in JavaScript 1.8. Generator expressions are a convenient, declarative form for creating generators with a syntax based on array comprehensions. Generator expressions also provide a convenient refactoring pattern, making it easy to switch between eager and on-demand generation of values in a sequence simply by changing the bracketing.

Examples

Extracting pages on demand from an array of URL's:

```
(xhrGet(url) for (url in getURLs()))
```

Filtering a sequence:

```
(x for (x in generateValues()) if (x.color === 'blue'))
```

Lazy cartesian product

```
(xhrGet(row, column) for (row in rows()) for (column in columns()))
```

Syntax

```
PrimaryExpression ::= ...
                    |  "(" Expression ("for" "(" LHSExpression "in" Expression")")+ ("if" "(" Expression ")")? ")"
```

Translation

A generator expression:

(*Expression*$_0$ **for (** *LHSExpression*$_1$ **in** *Expression*$_1$ **)** … **for (** *LHSExpression*$_n$ **) if (** *Expression* **)**$_{opt}$ **)**

can be defined by expansion to the expression:

(function () {
    for (let *LHSExpression*$_1$ **in** *Expression*$_1$ **) {**
        ...
        for (let *LHSExpression*$_n$ **in** *Expression*$_n$ **) {**
            if (*Expression* **)**$_{opt}$
                yield (*Expression*$_0$**);**
            }
        }
    }
})()

Notes

Background motivation for the syntactic sugar afforded by generator expressions:

- Peter Norvig's Sudoku solver based on constraint propagation, written in Python
- My port of Peter's solver to JS1.8

The critical uses of generator expressions, e.g., the actual parameter to `all` in:

```
    if (all(eliminate(values, s, d2) for (d2 in values[s]) if (d2 != d)))
        return values;
```

can only be desugared to generation function applications or an equivalent lazy iterator construct. They cannot be replaced with array comprehensions or any such eager construct without the solver taking exponential time and space creating eagerly populated arrays where it would have stopped early using lazy generator expressions, thanks to constraint propagation. Note how `all` is defined:

```
function all(seq) {
    for (let e in seq)
        if (!e)
            return false;
    return true;
}
```

so as to stop as soon as a value in the iterated sequence is falsy.

The JS1.8 version has some XXX comments and helper functions that show where methods such as the Array extras (`Array.prototype.every` instead of the custom `all` shown above, e.g.) are not iterator-friendly. This suggests the need for more generic methods that abstract over arrays and iterators.

— Brendan Eich 2010/06/27 19:47

strawman/generator_expressions.txt · Last modified: 2010/06/27 19:59 by brendan