```diff
diff --git a/../narcissus/lib/jsexec.js b/zaphod/chrome/content/narcissus/jsexec.js
index 163bd83..21d56ba 100644
--- a/../narcissus/lib/jsexec.js
+++ b/zaphod/chrome/content/narcissus/jsexec.js
@@ -57,1427 +57,1634 @@
 Narcissus.interpreter = (function() {

     var parser = Narcissus.parser;
     var definitions = Narcissus.definitions;
     var resolver = Narcissus.resolver;
     var hostGlobal = Narcissus.hostGlobal;
     var desugaring = Narcissus.desugaring;

+    // Faceted Value utilities
+    var FacetedValue = Zaphod.facets.FacetedValue;
+    var ProgramCounter = Zaphod.facets.ProgramCounter;
+    var Label = Zaphod.facets.Label;
+    var buildVal = Zaphod.facets.buildVal;
+    var evaluateEach = Zaphod.facets.evaluateEach;
+    var evaluateEachPair = Zaphod.facets.evaluateEachPair;
+    var strip = Zaphod.facets.strip;
+    var rebuild = Zaphod.facets.rebuild;
+
     // Set constants in the local scope.
     eval(definitions.consts);

     const StringMap = definitions.StringMap;
     const ObjectMap = definitions.ObjectMap;
     const StaticEnv = resolver.StaticEnv;
     const Def = resolver.Def;

     const GLOBAL_CODE = 0, EVAL_CODE = 1, FUNCTION_CODE = 2, MODULE_CODE = 3;

     // Control flow signals
     const BREAK_SIGNAL = {},
           CONTINUE_SIGNAL = {},
           RETURN_SIGNAL = {},
           END_SIGNAL = {};

     function isSignal(s) {
         if (s === BREAK_SIGNAL) return true;
         if (s === CONTINUE_SIGNAL) return true;
         if (s === RETURN_SIGNAL) return true;
         if (s === END_SIGNAL) return true;
         return false;
     }

-    function ExecutionContext(type, version) {
+    function ExecutionContext(type, pc, version) {
         this.type = type;
         this.version = version;
         // In Harmony, the global scope record is not exposed to the program.
         if (type === GLOBAL_CODE && version === "harmony") {
             this.scope = {object: globalScope, parent: null};
             this.thisObject = globalMirror;
         }
+        this.pc = pc;
     }

     function isStackOverflow(e) {
         var re = /InternalError: (script stack space quota is exhausted|too much recursion)/;
         return re.test(e.toString());
     }

+    function getPC() {
+        var x = ExecutionContext.current;
+        return x ? x.pc : new ProgramCounter();;
+    }
+
     // The underlying global object for narcissus.
     var globalBase = {
         // Value properties.
         NaN: NaN, Infinity: Infinity, undefined: undefined,

         // Function properties.
         eval: function eval(s) {
             if (typeof s !== "string")
                 return s;

             var x = ExecutionContext.current;
-            var x2 = new ExecutionContext(EVAL_CODE, x.version);
+            var x2 = new ExecutionContext(EVAL_CODE, x.pc, x.version);
             x2.thisObject = x.thisObject;
             x2.thisModule = x.thisModule;
             x2.caller = x.caller;
             x2.callee = x.callee;
             x2.scope = x.version === "harmony" ? { object: new Object, parent: x.scope } : x.scope;

             var ast = parser.parse(s);
             if (x.version === "harmony") {
                 resolver.resolve(ast, new StaticEnv(x.staticEnv));
                 instantiateModules(ast, x2.scope);
             }
             x2.execute(ast);
             return x2.result;
         },

+        // Displays only high alerts (assumes a simple hi/lo lattice
+        alert: function(msg){
+            let pc = getPC();
+            if (pc.containsStr('h') || pc.isEmpty())
+                alert(msg);
+            else
+                Zaphod.log('Suppressed unauthorized alert pc:' + pc + ' msg: "' + msg + '"');
+        },
+
+        exportValue: function(fv) {
+            let v = (fv instanceof FacetedValue) ? fv.unauthorized : fv;
+            alert('Attacker sees "' + v + '"');
+        },
+
         // Class constructors.  Where ECMA-262 requires C.length === 1, we declare
         // a dummy formal parameter.
         Function: function Function(dummy) {
             var p = "", b = "", n = arguments.length;
             if (n) {
                 var m = n - 1;
                 if (m) {
                     p += arguments[0];
                     for (var k = 1; k < m; k++)
                         p += "," + arguments[k];
                 }
                 b += arguments[m];
             }

             // XXX We want to pass a good file and line to the tokenizer.
             // Note the anonymous name to maintain parity with Spidermonkey.
             var t = new parser.Tokenizer("anonymous(" + p + ") {" + b + "}");

             // NB: Use the STATEMENT_FORM constant since we don't want to push this
             // function onto the fake compilation context.
             var f = parser.FunctionDefinition(t, null, false, parser.STATEMENT_FORM);
             var s = {object: global, parent: null};
             return newFunction(f,{scope:s});
```

```
        },
        Array: function (dummy) {
            // Array when called as a function acts as a constructor.
            return Array.apply(this, arguments);
        },
        String: function String(s) {
-           // Called as function or constructor: convert argument to string type.
-           s = arguments.length ? "" + s : "";
+           var argSpecified = arguments.length;
+           var newStr = evaluateEach(s, function(s,x) {
+               // Called as function or constructor: convert argument to string type.
+               return (argSpecified ? "" + s : "");
+           }, ExecutionContext.current);
+
            if (this instanceof String) {
                // Called as constructor: save the argument as the string value
                // of this String object and return this object.
-               this.value = s;
+               this.value = newStr;
+               var strlen = evaluateEach(newStr, function(s,x) {
+                   // Called as function or constructor: convert argument to string type.
+                   return s ? s.length : 0;
+               }, ExecutionContext.current);
+               definitions.defineProperty(this, 'length', strlen, true,
+                       true, true);
                return this;
            }
-           return s;
+           else return newStr;
        },

        // Don't want to proxy RegExp or some features won't work
        RegExp: RegExp,

        // Extensions to ECMA.
        load: function load(s) {
            if (typeof s !== "string")
                return s;

            evaluate(snarf(s), s, 1)
        },
        version: function() { return ExecutionContext.current.version; },
        quit: function() { throw END_SIGNAL; },
        assertEq: function() {
            return assertEq.apply(null, arguments);
-       }
+       },
+       cloak: function(v) {
+           // In Zaphod, sticking with a 2-element lattice
+           return Zaphod.facets.cloak(v,'h');
+       },
+       isFacetedValue: function(v) {
+           return (v instanceof FacetedValue);
+       },
+       // A view is represented as a program counter,
+       // except that all labels can only be 'positive'.
+       // If a label is not explicitly in the view,
+       // the viewer sees the unauthorized view.
+       getView: Zaphod.facets.getView,
+       getAuth: function(v) {
+           return Zaphod.facets.getView(v,
+                   new ProgramCounter(new Label('h')));
+       },
+       getUnAuth: function(v) {
+           return Zaphod.facets.getView(v,
+                   new ProgramCounter((new Label('h')).reverse()));
+       },
     };

+    // Load missing functions onto Array and String
+    ["concat", "every", "foreach", "isArray", "join", "map", "push", "pop",
+        "reverse", "reduce", "shift", "slice", "sort", "splice",
+        "toLocalString", "unshift"].forEach(function(fName) {
+            definitions.defineProperty(globalBase.Array, fName, Array[fName], false,
+                false, true);
+        });
+    //["charAt", "charCodeAt", "concat", "fromCharCode", "indexOf",
+    ["concat", "indexOf",
+        "lastIndexOf", "localeCompare", "match", "replace", "search", "slice",
+        "split", "substring", "toLowerCase", "toUpperCase", "trim", "valueOf",
+        //HTML methods
+        "big", "blink", "bold", "fixed", "fontcolor", "fontsize", "italics",
+        "link", "small", "strike", "sub", "sup"].forEach(function(fName) {
+            definitions.defineProperty(globalBase.String, fName, String[fName], false,
+                false, true);
+        });
+    var oldFCC = String.fromCharCode;
+    globalBase.String.fromCharCode = function(v1,v2) {
+        x = ExecutionContext.current;
+        return evaluateEachPair(v1, v2, function(v1,v2,x) {
+                if (v2) return oldFCC(v1,v2);
+                else return oldFCC(v1);
+        }, x);
+    };
+
+    // Operators
+    var ops = {};
+    ops[BITWISE_OR] = '|';
+    ops[BITWISE_XOR] = '^';
+    ops[BITWISE_AND] = '&';
+    ops[EQ] = '==';
+    ops[NE] = '!=';
+    ops[STRICT_EQ] = '===';
+    ops[STRICT_NE] = '!==';
+    ops[LT] = '<';
+    ops[LE] = '<=';
+    ops[GE] = '>=';
+    ops[GT] = '>';
+    ops[IN] = 'in';
+    ops[LSH] = '<<';
+    ops[RSH] = '>>';
+    ops[URSH] = '>>>';
+    ops[PLUS] = '+';
+    ops[MINUS] = '-';
+    ops[MUL] = '*';
+    ops[DIV] = '/';
+    ops[MOD] = '%';
+    ops[NOT] = '!';
+    ops[BITWISE_NOT] = '~';
+    ops[UNARY_PLUS] = '+';
+    ops[UNARY_MINUS] = '-';
+
+
+    function evalUnaryOp(c, x, op) {
+        var v = getValue(execute(c[0], x), x.pc);
+
+        return evaluateEach(v, function(v) {
+            return eval(ops[op] + "v");
+        }, x);
+    }
+
+    function evalBinOp(v1, v2, x, op) {
+        return evaluateEachPair(v1, v2, function(v1, v2) {
+            return eval('v1' + op + 'v2');
+          }, x);
+      }
+
    function wrapNative(name, val) {
        if (!definitions.isNativeCode(val))
            return val;
        return Proxy.createFunction(
            definitions.makePassthruHandler(val),
            function() { return val.apply(hostGlobal, arguments); },
            function() {
                var a = arguments;
                switch (a.length) {
                  case 0:
                    return new val();
                  case 1:
                    return new val(a[0]);
                  case 2:
                    return new val(a[0], a[1]);
                  case 3:
                    return new val(a[0], a[1], a[2]);
                  default:
                    var argStr = "";
                    for (var i = 0; i < a.length; i++)
                        argStr += 'a[' + i + '],';
                    return eval('new ' + name + '(' + argStr.slice(0,-1) + ');');
                }
            });
    }

    var hostHandler = definitions.blacklistHandler(hostGlobal,
        Narcissus.options.hiddenHostGlobals);
    var hostHandlerGet = hostHandler.get;
    hostHandler.get = function(receiver, name) {
        return wrapNative(name, hostHandlerGet(receiver, name));
    };
    var hostProxy = Proxy.create(hostHandler);

    var globalStaticEnv;                       // global static scope
    var moduleInstances = new ObjectMap();     // maps module instance objects -> module instances
    var global = Object.create(hostProxy, {}); // exposed global object (legacy)

    // unexposed global scope record (Harmony)
    var globalScope = Object.create(hostProxy, {});

    // exposed global scope mirror (Harmony)
    var globalMirror = Proxy.create(definitions.mirrorHandler(globalScope, true));

    function resetEnvironment() {
-        ExecutionContext.current = new ExecutionContext(GLOBAL_CODE, Narcissus.options.version);
+        ExecutionContext.current = new ExecutionContext(GLOBAL_CODE,
+                new ProgramCounter(), Narcissus.options.version);
        let names = Object.getOwnPropertyNames(global);
        for (let i = 0, n = names.length; i < n; i++) {
            delete global[names[i]];
        }
        for (let key in globalScope) {
            delete globalScope[key];
        }
        moduleInstances.clear();
        globalStaticEnv = new StaticEnv();

        let names = Object.getOwnPropertyNames(hostProxy);
        for (let i = 0, n = names.length; i < n; i++) {
            globalStaticEnv.bind(names[i], new Def());
        }
        for (let key in globalBase) {
            let val = globalBase[key];
            global[key] = val;
            globalScope[key] = val;
            // NB: this assumes globalBase never contains module or import bindings
            globalStaticEnv.bind(key, new Def());
        }
    }
    resetEnvironment();

    // Helper to avoid Object.prototype.hasOwnProperty polluting scope objects.
    function hasDirectProperty(o, p) {
        return Object.prototype.hasOwnProperty.call(o, p);
    }

    // Reflect a host class into the target global environment by delegation.
    function reflectClass(name, proto) {
        var gctor = global[name];
        definitions.defineProperty(gctor, "prototype", proto, true, true, true);
        definitions.defineProperty(proto, "constructor", gctor, false, false, true);
        return proto;
    }

    // Reflect Array -- note that all Array methods are generic.
    reflectClass('Array', new Array);

    // Reflect String, overriding non-generic methods.
    var gSp = reflectClass('String', new String);
    gSp.toSource = function () { return this.value.toSource(); };
    gSp.toString = function () { return this.value; };
    gSp.valueOf  = function () { return this.value; };
-   global.String.fromCharCode = String.fromCharCode;
+   //global.String.fromCharCode = String.fromCharCode;

    ExecutionContext.current = null;

    ExecutionContext.prototype = {
        caller: null,
        callee: null,
        scope: {object: global, parent: null},
        thisObject: global,
        thisModule: null,
        result: undefined,
        target: null,
        ecma3OnlyMode: false,

        // Execute a node in this execution context.
        execute: function(n) {
            var prev = ExecutionContext.current;
            ExecutionContext.current = this;
            try {
                execute(n, this);
            } finally {
                ExecutionContext.current = prev;
            }
        }
    };

    function Reference(base, propertyName, node) {
        this.base = base;
        this.propertyName = propertyName;
        this.node = node;
    }

    Reference.prototype.toString = function () { return this.node.getSource(); }

-   function getValue(v) {
+   function derefFacetedValue(v, pc) {
+       var k = v.label,
+               auth = v.authorized,
+               unauth = v.unauthorized;
+           if (pc.contains(k)) {
+               return getValue(auth, pc);
+           }
+           else if (pc.contains(k.reverse())) {
+               return getValue(unauth, pc);
+           }
+           else {
+               return buildVal(new ProgramCounter(k),
+                               getValue(auth, pc.join(k)),
+                               getValue(unauth, pc.join(k.reverse())));
+           }
+       }
+
+       function getValue(v, pc) {
+           if (v instanceof FacetedValue) {
+               return derefFacetedValue(v, pc);
+           }
            if (v instanceof Reference) {
                if (!v.base) {
                    // Hook needed for Zaphod
                    if (Narcissus.interpreter.getValueHook)
                        return Narcissus.interpreter.getValueHook(v.propertyName);
                    throw new ReferenceError(v.propertyName + " is not defined",
                                             v.node.filename, v.node.lineno);
                }
                return v.base[v.propertyName];
            }
            return v;
        }

-       function putValue(v, w, vn) {
-           if (v instanceof Reference)
-               return (v.base || global)[v.propertyName] = w;
+       function putValue(v, w, vn, pc) {
+           if (v instanceof FacetedValue) {
+               // x is not really an execution environment, but is being used a
+               // way of passing on data.
+               return evaluateEachPair(v, w, function(ref, val, x) {
+                   return putValue(ref, val, x.vn, x.pc);
+               }, {pc: pc, vn: vn});
+           }
+           else if (v instanceof Reference) {
+               //return (v.base || global)[v.propertyName] = w;
+               var base = v.base || global;
+               var oldVal = base[v.propertyName];
+               var newVal = base[v.propertyName] = buildVal(pc, w, oldVal);
+               // The returned value should be the local version, not the stored
+               // version.  Within a block, the extra labels are not needed and
+               // are simply wasteful.
+               return w;
+           }
            throw new ReferenceError("Invalid assignment left-hand side",
                                     vn.filename, vn.lineno);
        }

        function isPrimitive(v) {
            var t = typeof v;
            return (t === "object") ? v === null : t !== "function";
        }

        function isObject(v) {
            var t = typeof v;
            return (t === "object") ? v !== null : t === "function";
        }
```

```javascript
// If r instanceof Reference, v === getValue(r); else v === r.  If passed, rn
// is the node whose execute result was r.
function toObject(v, r, rn) {
    switch (typeof v) {
      case "boolean":
        return new global.Boolean(v);
      case "number":
        return new global.Number(v);
      case "string":
        return new global.String(v);
      case "function":
        return v;
      case "object":
        if (v !== null)
            return v;
    }
    var message = r + " (type " + (typeof v) + ") has no properties";
    throw rn ? new TypeError(message, rn.filename, rn.lineno)
             : new TypeError(message);
}

// reifyModule :: (Module) -> module instance object
function reifyModule(mod) {
    return mod.instance.proxy;
}

function bindImports(impDecls, x) {
    for (var i = 0; i < impDecls.length; i++) {
        var list = impDecls[i].pathList;
        for (var j = 0; j < list.length; j++) {
            bindImport(list[j], x);
        }
    }
}

function bindImport(decl, x) {
    var t = x.scope.object;
    var lhs = decl.children[0];
    var rhs = decl.children[1];
    var mod = lhs.denotedModule;

    function bind(importID, exportID) {
        definitions.defineGetter(t, importID, function() {
            var m = reifyModule(mod);
            return m[exportID];
        }, true);
    }

    if (rhs.type === IDENTIFIER) {
        if (rhs.value === "*") {
            mod.exports.forEach(function(exportID, exp) {
                if (!mod.exportedModules.has(exportID))
                    bind(exportID, exportID);
            });
        } else {
            bind(rhs.value, rhs.value);
        }
        return;
    }

    for (var i = 0; i < rhs.children.length; i++) {
        var pair = rhs.children[i];
        bind(pair.children[1].value, pair.children[0].value);
    }
}

function executeModule(n, x) {
        var m = x.scope.object[n.name];
        var inst = moduleInstances.get(m);
-       var x2 = new ExecutionContext(MODULE_CODE, x.version);
+       var x2 = new ExecutionContext(MODULE_CODE, x.pc, x.version);
        x2.scope = inst.scope;
        x2.thisObject = m;
        x2.thisModule = m;
        x2.execute(n.body);
        return m;
    }

    function execute(n, x) {
-       var a, c, f, i, j, r, s, t, u, v;
+       //try{
+       var a, c, f, i, j, r, s, t, u, v, v1, v2;
+
+       // Store the original pc
+       var pc = x.pc;

        switch (n.type) {
          case MODULE:
            if (n.body)
                x.result = executeModule(n, x);
            break;

          case IMPORT:
          case EXPORT:
            break;

          case FUNCTION:
            if (n.functionForm !== parser.DECLARED_FORM) {
                if (!n.name || n.functionForm === parser.STATEMENT_FORM) {
                    v = newFunction(n, x);
                    if (n.functionForm === parser.STATEMENT_FORM)
                        definitions.defineProperty(x.scope.object, n.name, v, true);
                } else {
                    t = new Object;
                    x.scope = {object: t, parent: x.scope};
                    try {
                        v = newFunction(n, x);
                        definitions.defineProperty(t, n.name, v, true, true);
                    } finally {
                        x.scope = x.scope.parent;
                    }
                }
            }
            break;

          case SCRIPT:
            t = x.scope.object;
            n.modAssns.forEach(function(name, node) {
                definitions.defineMemoGetter(t, name, function() {
                    return reifyModule(node.initializer.denotedModule);
                }, true);
            });
            bindImports(n.impDecls, x);
            a = n.funDecls;
            for (i = 0, j = a.length; i < j; i++) {
                s = a[i].name;
                f = newFunction(a[i], x);
                // ECMA-262 says variable bindings created by `eval' are deleteable.
                definitions.defineProperty(t, s, f, x.type !== EVAL_CODE);
            }
            a = n.varDecls;
            var defineVar;
            if (x.thisModule) {
                defineVar = function(obj, prop) {
```

```
                // start out as a getter/setter that throws on get
                definitions.defineGetterSetter(obj, prop, function() {
                    throw new ReferenceError(prop + " is not initialized");
                }, function(val) {
                    // on first set, replace with ordinary property
                    definitions.defineProperty(obj, prop, val, false);
                    return val;
                }, false);
            };
        } else {
            defineVar = function(obj, prop) {
                // ECMA-262 says variable bindings created by `eval' are deleteable.
                definitions.defineProperty(obj, prop, undefined, x.type !== EVAL_CODE, false);
            };
        }
        for (i = 0, j = a.length; i < j; i++) {
            u = a[i];
            s = u.name;
            if (u.readOnly && hasDirectProperty(t, s)) {
                throw new TypeError("Redeclaration of const " + s,
                                    u.filename, u.lineno);
            }
            if (u.readOnly || !hasDirectProperty(t, s)) {
                // Does not correctly handle 'const x;' -- see bug 592335.
                defineVar(t, s);
            }
        }
        // FALL THROUGH

      case BLOCK:
        c = n.children;
        for (i = 0, j = c.length; i < j; i++)
            execute(c[i], x);
        break;

      case IMPORT:
      case EXPORT:
        break;

      case IF:
-       if (getValue(execute(n.condition, x)))
+       let cond = getValue(execute(n.condition, x), pc);
+       if (cond instanceof FacetedValue) {
+           evaluateEach(cond, function(v, x) {
+               if (v)
+                   execute(n.thenPart, x);
+               else if (n.elsePart)
+                   execute(n.elsePart, x);
+           }, x);
+       }
+       else if (cond)
            execute(n.thenPart, x);
        else if (n.elsePart)
            execute(n.elsePart, x);
        break;

+     // FIXME: switch statement does not support faceted values
      case SWITCH:
-       s = getValue(execute(n.discriminant, x));
+       s = getValue(execute(n.discriminant, x), pc);
        a = n.cases;
        var matchDefault = false;
      switch_loop:
        for (i = 0, j = a.length; ; i++) {
            if (i === j) {
                if (n.defaultIndex >= 0) {
                    i = n.defaultIndex - 1; // no case matched, do default
                    matchDefault = true;
                    continue;
                }
                break;                      // no default, exit switch_loop
            }
            t = a[i];                       // next case (might be default!)
            if (t.type === CASE) {
-               u = getValue(execute(t.caseLabel, x));
+               u = getValue(execute(t.caseLabel, x), pc);
            } else {
                if (!matchDefault)          // not defaulting, skip for now
                    continue;
                u = s;                      // force match to do default
            }
            if (u === s) {
                for (;;) {                  // this loop exits switch_loop
                    if (t.statements.children.length) {
                        try {
                            execute(t.statements, x);
                        } catch (e if e === BREAK_SIGNAL && x.target === n) {
                            break switch_loop;
                        }
                    }
                    if (++i === j)
                        break switch_loop;
                    t = a[i];
                }
                // NOT REACHED
            }
        }
        break;

      case FOR:
-       n.setup && getValue(execute(n.setup, x));
+       n.setup && getValue(execute(n.setup, x), pc);
        // FALL THROUGH
      case WHILE:
-       while (!n.condition || getValue(execute(n.condition, x))) {
-           try {
-               execute(n.body, x);
-           } catch (e if e === BREAK_SIGNAL && x.target === n) {
-               break;
-           } catch (e if e === CONTINUE_SIGNAL && x.target === n) {
-               // Must run the update expression.
+       let whileCond = !n.condition || getValue(execute(n.condition, x), pc);
+       evaluateEach(whileCond, function(c,x) {
+           while (c) {
+               try {
+                   execute(n.body, x);
+               } catch (e if e === BREAK_SIGNAL && x.target === n) {
+                   break;
+               } catch (e if e === CONTINUE_SIGNAL && x.target === n) {
+                   // Must run the update expression.
+               }
+               n.update && getValue(execute(n.update, x), x.pc);
+               // FIXME: Label might become more secure over time.
+               c = !n.condition || getValue(execute(n.condition, x), x.pc);
+               if (c instanceof FacetedValue)
+                   throw new Error('Unhandled case: condition became more secure');
            }
-           n.update && getValue(execute(n.update, x));
-       }
+       }, x);
        break;

      case FOR_IN:
        u = n.varDecl;
```

```
            if (u)
                execute(u, x);
            r = n.iterator;
            s = execute(n.object, x);
-           v = getValue(s);
+           v = getValue(s, pc);

            // ECMA deviation to track extant browser JS implementation behavior.
            t = ((v === null || v === undefined) && !x.ecma3OnlyMode)
                ? v
                : toObject(v, s, n.object);
            a = [];
            for (i in t)
                a.push(i);
            for (i = 0, j = a.length; i < j; i++) {
-               putValue(execute(r, x), a[i], r);
+               putValue(execute(r, x), a[i], r, x.pc);
                try {
                    execute(n.body, x);
                } catch (e if e === BREAK_SIGNAL && x.target === n) {
                    break;
                } catch (e if e === CONTINUE_SIGNAL && x.target === n) {
                    continue;
                }
            }
            break;

          case DO:
-           do {
-               try {
-                   execute(n.body, x);
-               } catch (e if e === BREAK_SIGNAL && x.target === n) {
-                   break;
-               } catch (e if e === CONTINUE_SIGNAL && x.target === n) {
-                   continue;
-               }
-           } while (getValue(execute(n.condition, x)));
+           let doWhileCond = !n.condition || getValue(execute(n.condition, x), pc);
+           evaluateEach(doWhileCond, function(c,x) {
+               do {
+                   try {
+                       execute(n.body, x);
+                   } catch (e if e === BREAK_SIGNAL && x.target === n) {
+                       break;
+                   } catch (e if e === CONTINUE_SIGNAL && x.target === n) {
+                       // Must run the update expression.
+                   }
+                   // FIXME: Label might become more secure over time.
+                   c = !n.condition || getValue(execute(n.condition, x), x.pc);
+                   if (c instanceof FacetedValue)
+                       throw new Error('Unhandled case: condition became more secure');
+               } while(c);
+           }, x);
            break;

          case BREAK:
            x.target = n.target;
            throw BREAK_SIGNAL;

          case CONTINUE:
            x.target = n.target;
            throw CONTINUE_SIGNAL;

          case TRY:
            try {
                execute(n.tryBlock, x);
            } catch (e if !isSignal(e) && (j = n.catchClauses.length)) {
                x.result = undefined;
                for (i = 0; ; i++) {
                    if (i === j) {
                        throw e;
                    }
                    t = n.catchClauses[i];
                    x.scope = {object: {}, parent: x.scope};
                    definitions.defineProperty(x.scope.object, t.varName, e, true);
                    try {
-                       if (t.guard && !getValue(execute(t.guard, x)))
+                       if (t.guard && !getValue(execute(t.guard, x), pc))
                            continue;
                        execute(t.block, x);
                        break;
                    } finally {
                        x.scope = x.scope.parent;
                    }
                }
            } finally {
                if (n.finallyBlock)
                    execute(n.finallyBlock, x);
            }
            break;

          case THROW:
-           throw getValue(execute(n.exception, x));
+           throw getValue(execute(n.exception, x), pc);

          case RETURN:
            // Check for returns with no return value
-           x.result = n.value ? getValue(execute(n.value, x)) : undefined;
+           x.result = n.value ? getValue(execute(n.value, x), pc) : undefined;
            throw RETURN_SIGNAL;

          case WITH:
            r = execute(n.object, x);
-           t = toObject(getValue(r), r, n.object);
-           x.scope = {object: t, parent: x.scope};
-           try {
-               execute(n.body, x);
-           } finally {
-               x.scope = x.scope.parent;
-           }
+           t = getValue(r,pc);
+           evaluateEach(t, function(t,x) {
+               let o = toObject(t, r, n.object);
+               x.scope = {object: o, parent: x.scope};
+               try {
+                   execute(n.body, x);
+               } finally {
+                   x.scope = x.scope.parent;
+               }
+           }, x);
            break;

          case VAR:
          case CONST:
+           //FIXME: Real destructuring will be done by jsdesugar.js
+           function initializeVar(x, varName, varValue, type) {
+               var s;
+               let bv = buildVal(x.pc, varValue, undefined);
+               for (s = x.scope; s; s = s.parent) {
+                   if (hasDirectProperty(s.object, varName))
+                       break;
+               }
+               if (type === CONST)
+                   definitions.defineProperty(s.object, varName, bv, x.type !== EVAL_CODE, true);
+               else
+                   s.object[varName] = bv;
+           }
+
            c = n.children;
-           for (i = 0, j = c.length; i < j; i++) {
+           // destructuring assignments
+           if (c[0].name && c[0].name.type === ARRAY_INIT) {
+               let init = c[0].initializer;
+               if (init.type === ARRAY_INIT) {
+                   let initializers = init.children;
+                   for (i = 0, j = initializers.length; i < j; i++) {
+                       u = initializers[i];
+                       t = c[0].name.children[i].value;
+                       initializeVar(x, t, getValue(execute(u,x),pc), n.type);
+                   }
+               }
+               else {
+                   let arrVal = getValue(execute(init,x), pc);
+                   for (i = 0, j = arrVal.length; i < j; i++) {
+                       t = c[0].name.children[i].value;
+                       initializeVar(x, t, arrVal[i], n.type);
+                   }
+               }
+           }
+           else for (i = 0, j = c.length; i < j; i++) {
                u = c[i].initializer;
                if (!u)
                    continue;
                t = c[i].name;
-               for (s = x.scope; s; s = s.parent) {
-                   if (hasDirectProperty(s.object, t))
-                       break;
-               }
-               u = getValue(execute(u, x));
-               if (n.type === CONST)
-                   definitions.defineProperty(s.object, t, u, x.type !== EVAL_CODE, true);
-               else
-                   s.object[t] = u;
+               initializeVar(x, t, getValue(execute(u,x), pc), n.type);
            }
            break;

          case DEBUGGER:
            throw "NYI: " + definitions.tokens[n.type];

          case SEMICOLON:
            if (n.expression)
-               x.result = getValue(execute(n.expression, x));
+               x.result = getValue(execute(n.expression, x), pc);
            break;

          case LABEL:
            try {
                execute(n.statement, x);
            } catch (e if e === BREAK_SIGNAL && x.target === n.target) {
            }
            break;

          case COMMA:
            c = n.children;
            for (i = 0, j = c.length; i < j; i++)
-               v = getValue(execute(c[i], x));
+               v = getValue(execute(c[i], x), pc);
            break;

          case ASSIGN:
            c = n.children;
            r = execute(c[0], x);
            t = n.assignOp;
            if (t)
-               u = getValue(r);
-           v = getValue(execute(c[1], x));
+               u = getValue(r, x.pc);
+           v = getValue(execute(c[1], x), x.pc);
            if (t) {
-               switch (t) {
-                 case BITWISE_OR:  v = u | v; break;
-                 case BITWISE_XOR: v = u ^ v; break;
-                 case BITWISE_AND: v = u & v; break;
-                 case LSH:         v = u << v; break;
-                 case RSH:         v = u >> v; break;
-                 case URSH:        v = u >>> v; break;
-                 case PLUS:        v = u + v; break;
-                 case MINUS:       v = u - v; break;
-                 case MUL:         v = u * v; break;
-                 case DIV:         v = u / v; break;
-                 case MOD:         v = u % v; break;
-               }
+               v = evalBinOp(u, v, x, ops[t])
            }
-           putValue(r, v, c[0]);
+           putValue(r, v, c[0], x.pc);
            break;

          case HOOK:
            c = n.children;
-           v = getValue(execute(c[0], x)) ? getValue(execute(c[1], x))
-                                          : getValue(execute(c[2], x));
+           t = getValue(execute(c[0], x), pc);
+           v = evaluateEach(t, function(t,x) {
+               return t ? getValue(execute(c[1], x), x.pc)
+                        : getValue(execute(c[2], x), x.pc);
+           }, x);
            break;

          case OR:
            c = n.children;
-           v = getValue(execute(c[0], x)) || getValue(execute(c[1], x));
+           v = getValue(execute(c[0], x), pc);
+           if (v instanceof FacetedValue) {
+               let v2Thunk = function(pc) {
+                   return getValue(execute(c[1],x), pc);
+               };
+               v = evaluateEach(v, function(v1, x) {
+                   return v1 || v2Thunk(x.pc);
+               }, x);
+           }
+           else if (!v) {
+               v = getValue(execute(c[1], x), x.pc);
+           }
            break;

          case AND:
            c = n.children;
-           v = getValue(execute(c[0], x)) && getValue(execute(c[1], x));
+           v = getValue(execute(c[0], x), pc);
+           if (v instanceof FacetedValue) {
+               let v2Thunk = function(pc) {
+                   return getValue(execute(c[1],x), pc);
+               };
+               v = evaluateEach(v, function(v1, x) {
+                   return v1 && v2Thunk(x.pc);
+               }, x);
+               }
+               else if (v) {
+                   v = getValue(execute(c[1], x), x.pc);
+               }
                break;

              case BITWISE_OR:
-               c = n.children;
-               v = getValue(execute(c[0], x)) | getValue(execute(c[1], x));
-               break;
-
              case BITWISE_XOR:
-               c = n.children;
-               v = getValue(execute(c[0], x)) ^ getValue(execute(c[1], x));
-               break;
-
              case BITWISE_AND:
-               c = n.children;
-               v = getValue(execute(c[0], x)) & getValue(execute(c[1], x));
-               break;
-
              case EQ:
-               c = n.children;
-               v = getValue(execute(c[0], x)) == getValue(execute(c[1], x));
-               break;
-
              case NE:
-               c = n.children;
-               v = getValue(execute(c[0], x)) != getValue(execute(c[1], x));
-               break;
-
              case STRICT_EQ:
-               c = n.children;
-               v = getValue(execute(c[0], x)) === getValue(execute(c[1], x));
-               break;
-
              case STRICT_NE:
-               c = n.children;
-               v = getValue(execute(c[0], x)) !== getValue(execute(c[1], x));
-               break;
-
              case LT:
-               c = n.children;
-               v = getValue(execute(c[0], x)) < getValue(execute(c[1], x));
-               break;
-
              case LE:
-               c = n.children;
-               v = getValue(execute(c[0], x)) <= getValue(execute(c[1], x));
-               break;
-
              case GE:
-               c = n.children;
-               v = getValue(execute(c[0], x)) >= getValue(execute(c[1], x));
-               break;
-
              case GT:
-               c = n.children;
-               v = getValue(execute(c[0], x)) > getValue(execute(c[1], x));
-               break;
-
              case IN:
-               c = n.children;
-               v = getValue(execute(c[0], x)) in getValue(execute(c[1], x));
-               break;
-
-             case INSTANCEOF:
```

```
-               c = n.children;
-               t = getValue(execute(c[0], x));
-               u = getValue(execute(c[1], x));
-               if (isObject(u) && typeof u.__hasInstance__ === "function")
-                   v = u.__hasInstance__(t);
-               else
-                   v = t instanceof u;
-               break;
-
              case LSH:
-               c = n.children;
-               v = getValue(execute(c[0], x)) << getValue(execute(c[1], x));
-               break;
-
              case RSH:
-               c = n.children;
-               v = getValue(execute(c[0], x)) >> getValue(execute(c[1], x));
-               break;
-
              case URSH:
-               c = n.children;
-               v = getValue(execute(c[0], x)) >>> getValue(execute(c[1], x));
-               break;
-
              case PLUS:
-               c = n.children;
-               v = getValue(execute(c[0], x)) + getValue(execute(c[1], x));
-               break;
-
              case MINUS:
-               c = n.children;
-               v = getValue(execute(c[0], x)) - getValue(execute(c[1], x));
-               break;
-
              case MUL:
-               c = n.children;
-               v = getValue(execute(c[0], x)) * getValue(execute(c[1], x));
-               break;
-
              case DIV:
+             case MOD:
                c = n.children;
-               v = getValue(execute(c[0], x)) / getValue(execute(c[1], x));
+               v1 = getValue(execute(c[0], x), pc);
+               v2 = getValue(execute(c[1], x), pc);
+               v = evalBinOp(v1, v2, x, ops[n.type]);
                break;

-             case MOD:
+             case INSTANCEOF:
                c = n.children;
-               v = getValue(execute(c[0], x)) % getValue(execute(c[1], x));
+               t = getValue(execute(c[0], x), pc);
+               u = getValue(execute(c[1], x), pc);
+               v = evaluateEachPair(t, u, function(t, u, pc) {
+                   if (isObject(u) && typeof u.__hasInstance__ === "function")
+                       return u.__hasInstance__(t);
+                   else
+                       return t instanceof u;
+               }, x);
                break;

              case DELETE:
                t = execute(n.children[0], x);
-               v = !(t instanceof Reference) || delete t.base[t.propertyName];
+               v = evaluateEach(t, function(t,x) {
+                   return !(t instanceof Reference) || delete t.base[t.propertyName];
```

```diff
+             }, x);
              break;

          case VOID:
-             getValue(execute(n.children[0], x));
+             getValue(execute(n.children[0], x), pc);
              break;

          case TYPEOF:
              t = execute(n.children[0], x);
-             if (t instanceof Reference)
-                 t = t.base ? t.base[t.propertyName] : undefined;
-             v = typeof t;
+             v = evaluateEach(t, function(t,x) {
+                 if (t instanceof Reference)
+                     t = t.base ? t.base[t.propertyName] : undefined;
+                 return typeof t;
+             }, x);
              break;

          case NOT:
-             v = !getValue(execute(n.children[0], x));
-             break;
-
          case BITWISE_NOT:
-             v = ~getValue(execute(n.children[0], x));
-             break;
-
          case UNARY_PLUS:
-             v = +getValue(execute(n.children[0], x));
-             break;
-
          case UNARY_MINUS:
-             v = -getValue(execute(n.children[0], x));
+             c = n.children;
+             v = evalUnaryOp(c, x, n.type);
              break;

          case INCREMENT:
          case DECREMENT:
              t = execute(n.children[0], x);
-             u = Number(getValue(t));
+             u = getValue(t, pc);
              if (n.postfix)
                  v = u;
-             putValue(t, (n.type === INCREMENT) ? ++u : --u, n.children[0]);
+             u = evaluateEach(u, function(u,x) {
+                 let newVal = Number(n.type===INCREMENT ? u+1 : u-1);
+                 return putValue(t, newVal, n.children[0], x.pc);
+             }, x);
              if (!n.postfix)
                  v = u;
              break;

          case DOT:
              c = n.children;
              r = execute(c[0], x);
-             t = getValue(r);
-             u = c[1].value;
-             v = new Reference(toObject(t, r, c[0]), u, n);
+             t = getValue(r, pc);
+             v = evaluateEach(t, function(t,x) {
+                 u = c[1].value;
+                 if (u==='charAt') {
+                     this.THA = true;
+                 }
+                 return new Reference(toObject(t, r, c[0]), u, n);
+             }, x);
              break;

          case INDEX:
            c = n.children;
            r = execute(c[0], x);
-           t = getValue(r);
-           u = getValue(execute(c[1], x));
-           v = new Reference(toObject(t, r, c[0]), String(u), n);
+           t = getValue(r, pc);
+           u = getValue(execute(c[1], x), pc);
+           v = evaluateEachPair(t, u, function(t, u) {
+               return new Reference(toObject(t, r, c[0]), String(u), n);
+           }, x);
            break;

          case LIST:
            // Curse ECMA for specifying that arguments is not an Array object!
            v = {};
            c = n.children;
            for (i = 0, j = c.length; i < j; i++) {
-               u = getValue(execute(c[i], x));
+               u = getValue(execute(c[i], x), pc);
                definitions.defineProperty(v, i, u, false, false, true);
            }
            definitions.defineProperty(v, "length", i, false, false, true);
            break;

          case CALL:
            c = n.children;
            r = execute(c[0], x);
            a = execute(c[1], x);
-           f = getValue(r);
-           x.staticEnv = n.staticEnv;
-           if (isPrimitive(f) || typeof f.__call__ !== "function") {
-               throw new TypeError(r + " is not callable", c[0].filename, c[0].lineno);
-           }
-           t = (r instanceof Reference) ? r.base : null;
-           if (t instanceof Activation)
-               t = null;
-           v = f.__call__(t, a, x);
+           f = getValue(r, pc);
+           //v = evaluateEach(f, function(f,x) {
+           v = evaluateEachPair(f, r, function(f, r, x) {
+               x.staticEnv = n.staticEnv;
+               if (isPrimitive(f) || typeof f.__call__ !== "function") {
+                   throw new TypeError(r + " is not callable", c[0].filename, c[0].lineno);
+               }
+               t = (r instanceof Reference) ? r.base : null;
+               if (t instanceof Activation)
+                   t = null;
+               return f.__call__(t, a, x);
+           }, x);
            break;

          case NEW:
          case NEW_WITH_ARGS:
            c = n.children;
            r = execute(c[0], x);
-           f = getValue(r);
+           f = getValue(r, pc);
            if (n.type === NEW) {
                a = {};
                definitions.defineProperty(a, "length", 0, false, false, true);
            } else {
                a = execute(c[1], x);
            }
```

```
-             if (isPrimitive(f) || typeof f.__construct__ !== "function") {
-                 throw new TypeError(r + " is not a constructor", c[0].filename, c[0].lineno);
-             }
-             v = f.__construct__(a, x);
+             v = evaluateEach(f, function(f,x) {
+                 if (isPrimitive(f) || typeof f.__construct__ !== "function") {
+                     throw new TypeError(r + " is not a constructor", c[0].filename, c[0].lineno);
+                 }
+                 return f.__construct__(a, x);
+             }, x);
              break;

            case ARRAY_INIT:
              v = [];
              c = n.children;
              for (i = 0, j = c.length; i < j; i++) {
                  if (c[i])
-                     v[i] = getValue(execute(c[i], x));
+                     v[i] = getValue(execute(c[i], x), pc);
              }
              v.length = j;
              break;

            case OBJECT_INIT:
              v = {};
              c = n.children;
              for (i = 0, j = c.length; i < j; i++) {
                  t = c[i];
                  if (t.type === PROPERTY_INIT) {
                      let c2 = t.children;
-                     v[c2[0].value] = getValue(execute(c2[1], x));
+                     v[c2[0].value] = getValue(execute(c2[1], x), pc);
                  } else {
                      f = newFunction(t, x);
                      u = (t.type === GETTER) ? '__defineGetter__'
                                              : '__defineSetter__';
                      v[u](t.name, thunk(f, x));
                  }
              }
              break;

            case NULL:
              v = null;
              break;

            case THIS:
              v = x.thisObject;
              break;

            case TRUE:
              v = true;
              break;

            case FALSE:
              v = false;
              break;

            case IDENTIFIER:
              for (s = x.scope; s; s = s.parent) {
                  if (n.value in s.object)
                      break;
              }
              v = new Reference(s && s.object, n.value, n);
              break;

            case NUMBER:
            case STRING:
```

```
          case REGEXP:
            v = n.value;
            break;

          case GROUP:
            v = execute(n.children[0], x);
            break;

          default:
            throw "PANIC: unknown operation " + n.type + ": " + uneval(n);
        }

+        // For some odd reasons, faceted values sometimes forget their class.
+        // We rebuild them here if needed.
+        //v = rebuild(v);
         return v;
+        /*
+        } catch(e if !isSignal(e)) {
+            alert('Caught e: ' + e + ' \nn: ' + n);
+            throw END_SIGNAL;
+        }
+        */
     }

     function Activation(f, a) {
         for (var i = 0, j = f.params.length; i < j; i++)
             definitions.defineProperty(this, f.params[i], a[i], true);
         definitions.defineProperty(this, "arguments", a, true);
     }

     // Null Activation.prototype's proto slot so that Object.prototype.* does not
     // pollute the scope of heavyweight functions.  Also delete its 'constructor'
     // property so that it doesn't pollute function scopes.

     Activation.prototype.__proto__ = null;
     delete Activation.prototype.constructor;

     function FunctionObject(node, scope) {
         this.node = node;
         this.scope = scope;
         definitions.defineProperty(this, "length", node.params.length, true, true, true);
         var proto = {};
-        definitions.defineProperty(this, "prototype", proto, true);
+        //FIXME: should be read only, but this was causing some problems in dom.js.
+        //definitions.defineProperty(this, "prototype", proto, true);
+        definitions.defineProperty(this, "prototype", proto);
         definitions.defineProperty(proto, "constructor", this, false, false, true);
     }

     /*
      * ModuleInstance :: (Module, scope) -> ModuleInstance
      *
      * Dynamic semantic representation of a module.
      */
     function ModuleInstance(mod, scope) {
         this.module = mod;
         this.scope = scope;
     }

     /*
      * newModule :: (Module, scope) -> module instance object
      *
      * Instantiates a module node, producing a module instance object.
      */
     function newModule(mod, scope) {
         var exports = mod.exports;
```

```
    // the module instance
    mod.instance = new ModuleInstance(mod, {object: new Object, parent: scope});

    function keys() {
        var result = [];
        exports.forEach(function(name, exp) {
            result.push(name);
        });
        return result;
    }

    function getExportDescriptor(name) {
        if (exports.has(name)) {
            var exp = exports.get(name);
            var inst = exp.resolved.module.instance;

            return {
                value: inst.scope.object[exp.resolved.internalID],
                writable: false,
                enumerable: true,
                configurable: true
            };
        }

        throw new ReferenceError("no such export: " + name);
    }

    function getExportValue(receiver, name) {
        return getExportDescriptor(name).value;
    }

    function hasExport(name) {
        return exports.has(name);
    }

    function refuse() { }

    // the module instance proxy
    var instObj = Proxy.create({
        getOwnPropertyDescriptor: getExportDescriptor,
        getPropertyDescriptor: getExportDescriptor,
        getOwnPropertyNames: keys,
        defineProperty: refuse,
        "delete": refuse,
        fix: refuse,
        has: hasExport,
        hasOwn: hasExport,
        get: getExportValue,
        set: refuse,
        enumerate: keys,
        keys: keys
    });

    // associate the instance with the instance proxy
    moduleInstances.set(instObj, mod.instance);
    mod.instance.proxy = instObj;

    return instObj;
}

function instantiateModules(n, scope) {
    n.modDefns.forEach(function(name, defn) {
        var m = defn.module;
        var instObj = newModule(m, scope);
        var inst = moduleInstances.get(instObj);
        definitions.defineProperty(scope.object, name, instObj, true, true);
        instantiateModules(m.node.body, inst.scope);
        });
    }

    function getPropertyDescriptor(obj, name) {
        while (obj) {
            if (({}).hasOwnProperty.call(obj, name))
                return Object.getOwnPropertyDescriptor(obj, name);
            obj = Object.getPrototypeOf(obj);
        }
    }

    function getOwnProperties(obj) {
        var map = {};
        for (var name in Object.getOwnPropertyNames(obj))
            map[name] = Object.getOwnPropertyDescriptor(obj, name);
        return map;
    }

    // Returns a new function wrapped with a Proxy.
    function newFunction(n, x) {
        var fobj = new FunctionObject(n, x.scope);
        var handler = definitions.makePassthruHandler(fobj);
        var p = Proxy.createFunction(handler,
                                     function() { return fobj.__call__(this, arguments, x); },
                                     function() { return fobj.__construct__(arguments, x); });
        return p;
    }

    var FOp = FunctionObject.prototype = {

        // Internal methods.
        __call__: function (t, a, x) {
-           var x2 = new ExecutionContext(FUNCTION_CODE, x.version);
+           var x2 = new ExecutionContext(FUNCTION_CODE, x.pc, x.version);
            x2.thisObject = t || global;
            x2.thisModule = null;
            x2.caller = x;
            x2.callee = this;
            definitions.defineProperty(a, "callee", this, false, false, true);
            var f = this.node;
            x2.scope = {object: new Activation(f, a), parent: this.scope};

            try {
                x2.execute(f.body);
            } catch (e if e === RETURN_SIGNAL) {
                return x2.result;
            }
            return undefined;
        },

        __construct__: function (a, x) {
            var o = new Object;
            var p = this.prototype;
            if (isObject(p))
                o.__proto__ = p;
            // else o.__proto__ defaulted to Object.prototype

            var v = this.__call__(o, a, x);
            if (isObject(v))
                return v;
            return o;
        },

        __hasInstance__: function (v) {
            if (isPrimitive(v))
                return false;
            var p = this.prototype;
            if (isPrimitive(p)) {
                throw new TypeError("'prototype' property is not an object",
                                    this.node.filename, this.node.lineno);
            }
            var o;
            while ((o = Object.getPrototypeOf(v))) {
                if (o === p)
                    return true;
                v = o;
            }
            return false;
        },

        // Standard methods.
        toString: function () {
            return this.node.getSource();
        },

        apply: function (t, a) {
            // Curse ECMA again!
            if (typeof this.__call__ !== "function") {
                throw new TypeError("Function.prototype.apply called on" +
                                    " uncallable object");
            }

            if (t === undefined || t === null)
                t = global;
            else if (typeof t !== "object")
                t = toObject(t, t);

            if (a === undefined || a === null) {
                a = {};
                definitions.defineProperty(a, "length", 0, false, false, true);
            } else if (a instanceof Array) {
                var v = {};
                for (var i = 0, j = a.length; i < j; i++)
                    definitions.defineProperty(v, i, a[i], false, false, true);
                definitions.defineProperty(v, "length", i, false, false, true);
                a = v;
            } else if (!(a instanceof Object)) {
                // XXX check for a non-arguments object
                throw new TypeError("Second argument to Function.prototype.apply" +
                                    " must be an array or arguments object",
                                    this.node.filename, this.node.lineno);
            }

            return this.__call__(t, a, ExecutionContext.current);
        },

        call: function (t) {
            // Curse ECMA a third time!
-           var a = Array.prototype.splice.call(arguments, 1);
+           //var a = Array.prototype.splice.call(arguments, 1);
+           var a = Array.prototype.slice.call(arguments, 1);
            return this.apply(t, a);
        }
    };

    // Connect Function.prototype and Function.prototype.constructor in global.
    reflectClass('Function', FOp);

    // Help native and host-scripted functions be like FunctionObjects.
    var Fp = Function.prototype;
    var REp = RegExp.prototype;

    if (!('__call__' in Fp)) {
        definitions.defineProperty(Fp, "__call__",
                                    function (t, a, x) {
                                        // Curse ECMA yet again!
-                                       a = Array.prototype.splice.call(a, 0, a.length);
-                                       return this.apply(t, a);
+                                       //FIXME: Need support for faceted arguments here
+                                       //a = Array.prototype.splice.call(a, 0, a.length);
+                                       a = Array.prototype.slice.call(a, 0, a.length);
+                                       if (!definitions.isNativeCode(this)) {
+                                           return this.apply(t, a);
+                                       }
+                                       var thisObj = this;
+                                       switch (a.length) {
+                                         case 1:
+                                            return evaluateEach(rebuild(a[0],x.pc), function(v,x) {
+                                               return thisObj.call(t, v);
+                                            }, x);
+                                         case 2:
+                                            return evaluateEachPair(strip(a[0],x.pc), strip(a[1],x.pc),
+                                                function(v1,v2,x) {
+                                                    return thisObj.call(t, v1, v2);
+                                                }, x);
+                                         //No support for more than 2 FV
+                                         //arguments for native functions
+                                         default:
+                                            return thisObj.apply(t, a);
+                                       }
                                    }, true, true, true);
         definitions.defineProperty(REp, "__call__",
                                    function (t, a, x) {
-                                       a = Array.prototype.splice.call(a, 0, a.length);
+                                       //a = Array.prototype.splice.call(a, 0, a.length);
+                                       a = Array.prototype.slice.call(a, 0, a.length);
                                        return this.exec.apply(this, a);
                                    }, true, true, true);
         definitions.defineProperty(Fp, "__construct__",
                                    function (a, x) {
-                                       a = Array.prototype.splice.call(a, 0, a.length);
+                                       //a = Array.prototype.splice.call(a, 0, a.length);
+                                       a = Array.prototype.slice.call(a, 0, a.length);
                                        switch (a.length) {
                                          case 0:
                                            return new this();
                                          case 1:
                                            return new this(a[0]);
                                          case 2:
                                            return new this(a[0], a[1]);
                                          case 3:
                                            return new this(a[0], a[1], a[2]);
                                          default:
                                            var argStr = "";
                                            for (var i=0; i<a.length; i++) {
                                                argStr += 'a[' + i + '],';
                                            }
                                            return eval('new this(' + argStr.slice(0,-1) + ');');
                                        }
                                    }, true, true, true);

         // Since we use native functions such as Date along with host ones such
         // as global.eval, we want both to be considered instances of the native
         // Function constructor.
         definitions.defineProperty(Fp, "__hasInstance__",
                                    function (v) {
                                        return v instanceof Function || v instanceof global.Function;
                                    }, true, true, true);
     }

     function thunk(f, x) {
         return function () { return f.__call__(this, arguments, x); };
     }

     function resolveGlobal(ast) {
         // clone the static env so we can rollback if compilation fails
         var extendedStaticEnv = globalStaticEnv.copy();
         resolver.resolve(ast, extendedStaticEnv);
         // compilation succeeded, so commit to the extended static env
         globalStaticEnv = extendedStaticEnv;
     }

     function evaluate(s, f, l) {
         if (typeof s !== "string")
             return s;

-       var x = new ExecutionContext(GLOBAL_CODE, Narcissus.options.version);
+       var x = new ExecutionContext(GLOBAL_CODE, new ProgramCounter(), Narcissus.options.version);
         var ast = parser.parse(s, f, l);
         if (Narcissus.options.desugarExtensions)
             ast = desugaring.desugar(ast);
         if (x.version === "harmony") {
             resolveGlobal(ast);
             instantiateModules(ast, x.scope);
         }
         x.execute(ast);
         return x.result;
     }

     function printStackTrace(stack) {
         var st = String(stack).split(/\n/);
         // beautify stack trace:
         //   - eliminate blank lines
         //   - sanitize confusing trace lines for getters and js -e expressions
         //   - simplify source location reporting
         //   - indent
         for (var i = 0; i < st.length; i++) {
             var line = st[i].trim();
             if (line) {
                 line = line.replace(/^(\(\))?@/, "<unknown>@");
                 line = line.replace(/@(.*\/|\\)?([^\/\\]+:[0-9]+)/, " at $2");
                 print("    in " + line);
             }
         }
     }

     // A read-eval-print-loop that roughly tracks the behavior of the js shell.
     function repl() {

         // Display a value similarly to the js shell.
         function display(x) {
             if (typeof x === "object") {
                 // At the js shell, objects with no |toSource| don't print.
                 if (x !== null && "toSource" in x) {
                     try {
                         print(x.toSource());
                     } catch (e) {
                     }
                 } else {
                     print("null");
                 }
             } else if (typeof x === "string") {
                 print(uneval(x));
             } else if (typeof x !== "undefined") {
                 // Since x must be primitive, String can't throw.
                 print(String(x));
             }
         }
```

```
         // String conversion that never throws.
         function string(x) {
             try {
                 return String(x);
             } catch (e) {
                 return "unknown (can't convert to string)";
             }
         }

         const BREAK_INTERACTION = {};

         // isCommand :: (string) -> boolean
         function isCommand(line) {
             switch (line.trim()) {
               case ".help":
                 print(".begin  Begin multiline input mode.");
                 print(".break  Sometimes you get stuck in a place you can't get out... This will get you out.");
                 print(".clear  Break, and also clear the global environment.");
                 print(".end    End multiline input mode.");
                 print(".exit   Exit the prompt.");
                 print(".help   Show repl options.");
                 return true;

               case ".clear":
                 resetEnvironment();
                 // FALL THROUGH

               case ".break":
                 throw BREAK_INTERACTION;

               case ".exit":
                 throw END_SIGNAL;
             }
             return false;
         }

-        var x = new ExecutionContext(GLOBAL_CODE, Narcissus.options.version);
+        var x = new ExecutionContext(GLOBAL_CODE, new ProgramCounter(), Narcissus.options.version);

         // Line number in/out parameter to parser.parseStdin.
         var ln = {value: 0};

         ExecutionContext.current = x;
         for (;;) {
             x.result = undefined;
             putstr("njs> ");
             var src = readline();

             // If readline receives EOF it returns null.
             if (src === null) {
                 print("");
                 break;
             }
             ++ln.value;

             try {
                 var ast = parser.parseStdin(src, ln, "...  ", isCommand);
                 if (Narcissus.options.desugarExtensions)
                     ast = desugaring.desugar(ast);
                 if (x.version === "harmony") {
                     resolveGlobal(ast);
                     instantiateModules(ast, x.scope);
                 }
                 execute(ast, x);
                 display(x.result);
             } catch (e if e === END_SIGNAL) {
                 break;
             } catch (e if e === BREAK_INTERACTION) {
                 continue;
             } catch (e if e instanceof SyntaxError) {
                 const PREFIX = (e.filename || "stdin") + ":" + e.lineNumber + ": ";
                 print(PREFIX + e.toString());
                 print(PREFIX + e.source);
                 print(PREFIX + ".".repeat(e.cursor) + "^");
             } catch (e if e instanceof Error) {
                 print((e.filename || "stdin") + ":" +  e.lineNumber + ": " + e.toString());
                 if (e.stack)
                     printStackTrace(e.stack);
             } catch (e) {
                 print("unexpected Narcissus exception (" + e + ")");
                 throw e;
             }
         }
         ExecutionContext.current = null;
     }

     function test(thunk) {
         try {
             thunk();
         } catch (e) {
             print(e.fileName + ":" + e.lineNumber + ": " + e.name + ": " + e.message);
             printStackTrace(e.stack);
             return false;
         }
         return true;
     }

     return {
         // resetEnvironment wipes any properties added externally to global,
         // but properties added to globalBase will persist.
         global: global,
         globalBase: globalBase,
         resetEnvironment: resetEnvironment,
         evaluate: evaluate,
         getValueHook: null,
         repl: repl,
-       test: test
+       test: test,
+       getPC: getPC
     };

 }());
```