Specifying JavaScript in ML

Dave Herman
Cormac Flanagan
JavaScript: the “j” in Ajax

http://www.ecmascript.org
JavaScript is not Java-Lite.

JavaScript is almost nothing like Java!

- Multi-paradigm
- Functional! Lexically-scoped!
- Prototypes, no classes (until now…)
- Dynamically typed (until now…)

http://www.ecmascript.org
Outline

I. How we got here
II. Specifying JavaScript with ML
III. The evolution of JavaScript
IV. Status report
I. Testimonial: coming to ML
Standardized JavaScript

- Many implementations exist (multiple browsers, JVM, Flash, ...)
- Highly competitive market
- First standardized at Ecma Int’l in ’97

http://www.ecmascript.org
Existing JavaScript specifications

Grammar production:

\[
\text{ConditionalExpression} \rightarrow \text{LogicalORExpression} \ ? \ \text{AssignmentExpression} \ : \ \text{AssignmentExpression}
\]

Evaluation:

1. Evaluate \text{LogicalORExpression}.
2. Call GetValue(Result(1)).
3. Call ToBoolean(Result(2)).
4. If Result(3) is \text{false}, go to step 8.
5. Evaluate the first \text{AssignmentExpression}.
6. Call GetValue(Result(5)).
7. Return Result(6).
8. Evaluate the second \text{AssignmentExpression}.
9. Call GetValue(Result(8)).
10. Return Result(9).

Any bugs in there? **Who knows?**
Existing JavaScript specifications

From: Brendan Eich  
Date: 12/2/2005  
To: Dave Herman  
Subject: Specifying JavaScript semantics

...In my view, we TG1’ers desperately need a machine-checkable specification... Others in TG1 may agree in principle, but are not wild about taking too much time to develop a checkable spec. So we are looking for help...
Meeting in the middle

- informal prose, pseudocode
- definitional interpreter
- formal methods, mechanized frameworks
- precise, accurate

http://www.ecmascript.org
One interpreter, several artifacts

- Read—literate source
- Run, test—executable
- Prove theorems—source code (maybe)
An executable spec in SML/NJ

- First-level sanity checks
- Tests: JS2 standard library, JS2 → Flash VM compiler
- Biggest benefit: Mozilla/Adobe regression test suites for JavaScript 1.x
II. Specifying JavaScript with SML
Interpreter style

- No gratuitous optimization
- No gratuitous mutation (duh!)
- Modularize language feature encodings (“keep the monster in a box”)
- Big-step, impure, direct style

```
eval : EXPR * ENV -> VAL
```
Exploiting ML: algebraic datatypes

datatype EXPR =
    UnaryExpr of (UNOP * EXPR)
  | BinaryExpr of (BINOP * EXPR * EXPR)
  | TernaryExpr of (EXPR * EXPR * EXPR)
  | LiteralExpr of LITERAL
  | CallExpr of { func: EXPR,
                actuals: EXPR list }
  | ...

and STMT =
    ExprStmt of EXPR
  | ForStmt of FOR_STMT
  | WhileStmt of WHILE_STMT
  | IfStmt of { cnd: EXPR,
              thn: STMT,
              els: STMT }
  | ...

http://www.ecmascript.org
Exploiting ML: refs

datatype VAL =
    Object of OBJ
  | Null
  | Undef
and OBJ =
    Obj of { ident: int,
      tag: VAL_TAG,
      proto: VAL,
      props: PROP_BINDINGS }
and VAL_TAG =
    ObjectTag of FIELD_TYPE list
  | ArrayTag of TYPE_EXPR list
  | FunctionTag of FUNC_TYPE
  | ClassTag of NAME

withtype PROP = { ty: TYPE_EXPR,
      attrs: ATTRS,
      state: VAL }
and PROP_BINDINGS = (NAME * PROP) list ref
Exploiting ML: exceptions

- Non-local jumps in JS: return, throw, tail calls
- Tail calls via modified trampoline:
  ```
  exception TailCallException of (unit -> VAL)
  ```
- Non-tail function call protocol:
  ```
  [[expr]]
  handle TailCallException thunk => thunk()
  ```
Exploiting (S)ML(/NJ): callcc

- JavaScript *generators* (coroutines)
- *yield* instead of *return*: suspend current procedure activation as an object

```javascript
function nats() {
    var i = 0;
    while (true) {
        yield i;
        i++;
    }
}
var gen = nats();
print(gen.next()); // 0
print(gen.next()); // 1
print(gen.next()); // 2
```
Exploiting (S)ML(/NJ): callcc

- `yield` captures a (delimited) continuation
- Alternatives: threads, delimited continuations, CPS
- Benefit of `callcc`: localizing the encoding

http://www.ecmascript.org
Exploiting ML: modules

datatype OBJ = Obj of { ... props: PROP_BINDINGS ... }
with type PROP = { ty: TYPE_EXPR,
  attrs: ATTRS,
  state: VAL }
and PROP_BINDINGS = (NAME * PROP) list ref
Exploiting ML: modules

datatype OBJ = Obj of { ... props: PROP_BINDINGS ... }

with type PROP = { ty: TYPE_EXPR,
    attrs: ATTRS,
    state: VAL }

and PROP_BINDINGS = (PROP NameMap.map) ref

structure NameMap = BinaryMapFn (NameKey);
(* structure NameMap = SplayMapFn (NameKey); *)
(* structure NameMap = ListMapFn (NameKey); *)
Limitations of code

- Underspecification: sort must be $n \log(n)$
- Overspecification: type(x) reveals concrete implementation of interface
- Conclusion: *prose is still necessary*
Frustrations with SML

- Non-nested `withtype` declarations
- Deciphering type inference errors
- No recursive modules
III. The evolution of JavaScript
Typed functional programming for the web

http://www.ecmascript.org
Dynamically typed

*Mostly* well-behaved scoping

Prototypes inspired by Self

Objects: (string × value) table
JavaScript 2.0/ECMAScript Edition 4

- “Opt-in” static typing
- Improved lexical scoping
- Prototypes and classes
- Objects: ((key × string) × value) table
- Generous syntactic sugar
- Advanced control constructs (generators)
Gradual typing

- Conservative extension of dynamically typed JavaScript 1.x
- Optional type annotations
- Optional static type checker
- Careful interface between typed and untyped code
Type system

- Nominal types (Java, C#)
- Structural types
  
  ```
  type thunk = function():int;
  function foo(f:thunk):[int] { ... }
  var obj : { m:function(thunk):[int] } = { m:foo };
  ```

- Type “dynamic”:
  
  ```
  var image : * = read(filename);
  switch type (image) { ... }
  ```

- Parameterized types
IV. Status report
Status report

- Feature freeze, coding and writing
- ~25 KLOC mostly pure SML, ~12 KLOC JS2 standard libraries
- SML/NJ, proof-of-concept ports to SML.NET and MLton
- Release: 2008 (tentatively)

http://www.ecmascript.org
Steal this source code

- Read, run, test ✓
- Prove theorems?
- Industry-scale case study for research
  - Experimental language extensions
  - Development tools, IDE’s
  - Inference, migration tools
  - ...what else? Have at it:
    www.ecmascript.org/download.php
Thank you!

http://www.ecmascript.org