Interpreting JavaScript

CS565

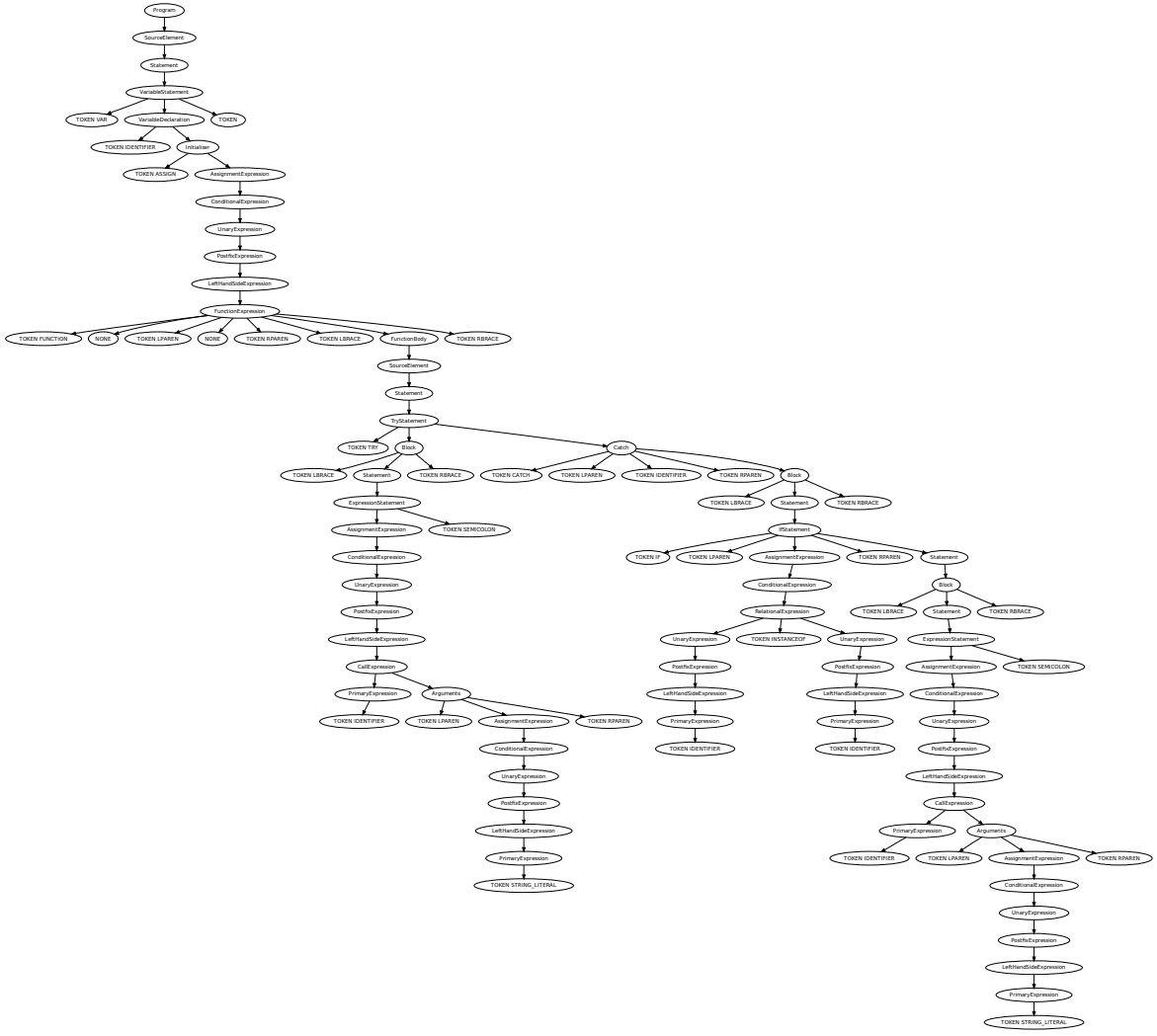
Purdue University

February 4, 2010

Code to abstract syntax tree

```
var f = function() {
    try {
        eval("throw␣new␣Object();");
    } catch (x) {
        if (x instanceof Object) {
            print("Object");
        }
    }
}
```

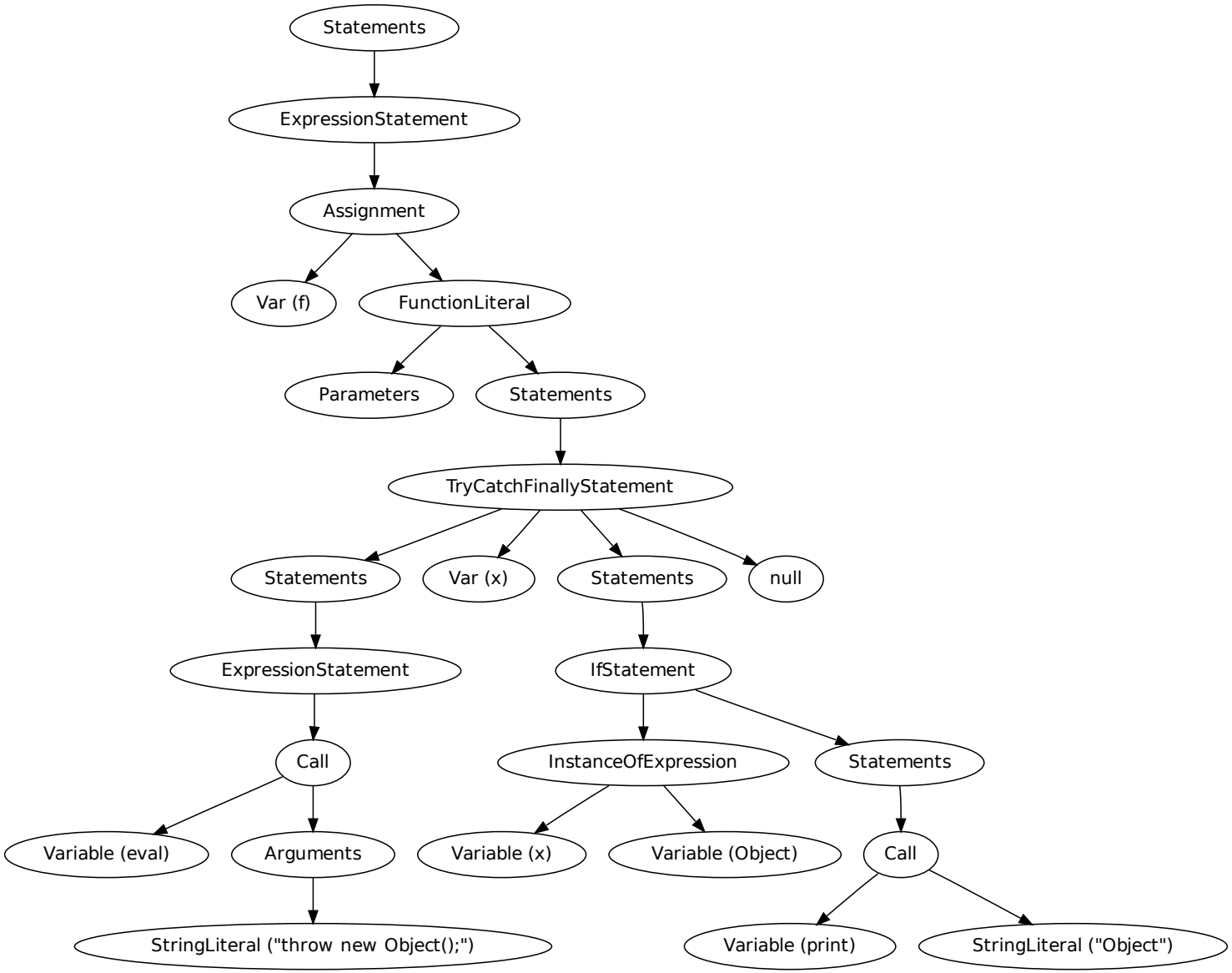
Parse tree


Program
SourceElement
Statement
VariableStatement
TOKEN VAR
VariableDeclaration
TOKEN
TOKEN IDENTIFIER
Initialiser
TOKEN ASSIGN
AssignmentExpression
ConditionalExpression
UnaryExpression
PostfixExpression
LeftHandSideExpression
FunctionExpression
TOKEN FUNCTION
NONE
TOKEN LPAREN
NONE
TOKEN RPAREN
TOKEN LBRACE
FunctionBody
TOKEN RBRACE
SourceElement
Statement
TryStatement
TOKEN TRY
Block
TOKEN LBRACE
Statement
TOKEN RBRACE
ExpressionStatement
AssignmentExpression
TOKEN SEMICOLON
ConditionalExpression
UnaryExpression
PostfixExpression
LeftHandSideExpression
CallExpression
PrimaryExpression
Arguments
TOKEN IDENTIFIER
TOKEN LPAREN
AssignmentExpression
TOKEN RPAREN
ConditionalExpression
UnaryExpression
PostfixExpression
LeftHandSideExpression
PrimaryExpression
TOKEN STRING_LITERAL
Catch
TOKEN CATCH
TOKEN LPAREN
TOKEN IDENTIFIER
TOKEN RPAREN
Block
TOKEN LBRACE
Statement
TOKEN RBRACE
IfStatement
TOKEN IF
TOKEN LPAREN
AssignmentExpression
TOKEN RPAREN
Statement
ConditionalExpression
RelationalExpression
UnaryExpression
TOKEN INSTANCEOF
UnaryExpression
PostfixExpression
LeftHandSideExpression
PrimaryExpression
TOKEN IDENTIFIER
PostfixExpression
LeftHandSideExpression
PrimaryExpression
TOKEN IDENTIFIER
Block
TOKEN LBRACE
Statement
TOKEN RBRACE
ExpressionStatement
AssignmentExpression
TOKEN SEMICOLON
ConditionalExpression
UnaryExpression
PostfixExpression
LeftHandSideExpression
CallExpression
PrimaryExpression
Arguments
TOKEN IDENTIFIER
TOKEN LPAREN
AssignmentExpression
TOKEN RPAREN
ConditionalExpression
UnaryExpression
PostfixExpression
LeftHandSideExpression
PrimaryExpression
TOKEN STRING_LITERAL

Parse tree to AST

- Parse trees include a lot of irrelevant nodes
- AST includes only what the interpreter needs
- Usually each AST node has a well-defined non-trivial behavior

Example AST


Statements
ExpressionStatement
Assignment
Var (f)
FunctionLiteral
Parameters
Statements
TryCatchFinallyStatement
Statements
Var (x)
Statements
null
ExpressionStatement
IfStatement
Call
InstanceOfExpression
Statements
Variable (eval)
Arguments
Variable (x)
Variable (Object)
Call
StringLiteral ("throw new Object();")
Variable (print)
StringLiteral ("Object")

Types of AST nodes

- Statement**s**
- Statement
  - ExpressionStatement
  - IfStatement
  - ...
- Expression
  - Assignment
  - Variable
  - ...
- Literal
  - StringLiteral
  - ArrayLiteral
  - ...

Statements

It is common to have an AST node to specify sequence. A Statement**s** node contains multiple Statement nodes to be executed.

Statement

An individual statement. As a statement, interpreting a statement node produces no value.
Statements such as try/catch/finally and throw will require implementing exceptions (as described last week).
Break and Continue are special cases: You will need to use a mechanism similar to exceptions to roll back to the WhileStatement (or ForStatement).

Expression

The basic block which defines the semantic behavior of a program.
Left-hand expression (e.g. Var from the example AST) will need to return *references*.
All other expressions return *values*

Literals

Simple to execute: Just turn the value in the AST into a JavaScript type.

Producing an AST

- Walk over your parse tree
- If a node is interesting, turn it into an AST node!
- Otherwise, just walk deeper
- Some parse nodes have multiple possible AST nodes (think multiplication expressions)

Producing an AST — Example