JS

Functions, Variables, and With in JavaScript

by Lee Fan Burke Allen Fetscher, John Morgan Greene, Dylan A Hirshkowitz, Daniel Joseph Lieberman, Josiah William Matlack, Ethan James Rombu, Maciej Swiech, Matthew Pierce Wampler-Doty, Kaicheng Zhang, and Robby Findler

This is a model of functions and variables and `with` in JavaScript. It demonstrates how scope objects work (assuming that our declarations have already been lifted to the nearest enclosing scope or the top of the program), showing `with` in its full glory. It includes a simplified version of objects (with normal fields and getter/setter pairs but not prototypes or other features of JavaScript's objects) — just enough to be able to explain scope objects.

Note that values in the language include only strings and objects: numbers are not JavaScript values but references to objects in the store.

MODULE JS-SYNTAX

SYNTAX $Block ::= \texttt{var}\ Ids\ ;\ Stmt$

SYNTAX $Stmt ::= Expr$
| $\texttt{return}\ Expr$ [strict]
| $Stmt\ ;\ Stmt$ [strict(1)]

SYNTAX $Expr ::= \{ObjField\ ;\ \ldots\}$ [strict]
| $Expr[Expr]$ [sequence]
| $Expr.Expr = Expr$ [sequence]
| $Id = Expr$ [strict(2)]
| Id
| $\texttt{function}\ (Ids)\{Block\}$
| $Expr(Exprs)$ [sequence]
| $\texttt{print}\ (Expr)$ [strict]
| $(Expr)$ [bracket]
| $Value$
| $\texttt{with}\ (Expr)\{Stmt\}$ [strict(1)]

SYNTAX $Exprs ::= List\{Expr, ","\}$ [sequence]

SYNTAX $Value ::= Ref$
| $String$
| $\texttt{undefined}$
| $\texttt{null}$

SYNTAX $Values ::= List\{Value, ","\}$

SYNTAX $Ref ::= Int$

SYNTAX $Refs ::= List\{Ref, ","\}$

SYNTAX $Ids ::= List\{Id, ","\}$

SYNTAX $ObjFields ::= List\{ObjField, ","\}$ [sequence]

SYNTAX $ObjField ::= String\ :\ Expr$ [strict(2)]
| $Getter$
| $Setter$

SYNTAX $Getter ::= \texttt{get}\ Id()\{Block\}$

SYNTAX $Setter ::= \texttt{set}\ Id(Id)\{Block\}$

END MODULE

MODULE JS

SYNTAX $ValObjField ::= Getter$
| $Setter$

SYNTAX $\mathit{isValObjField}(\_, K\ell) = \mathit{true}$

SYNTAX $ValObjFields ::= List\{ValObjField, ","\}$

SYNTAX $MaybeFuncObj ::= FuncObj$
| $\texttt{nofunobj}$

SYNTAX $FuncObj ::= \texttt{FuncObj}\ (Refs, Ids, Block)$

SYNTAX $AccessPair ::= MaybeFuncObj\ \texttt{++}\ MaybeFuncObj$

SYNTAX $K ::= \texttt{Obj}\ (Map, MaybeFuncObj)$

SYNTAX $KResult ::= Value$
| $ValObjField$
| $FuncObj$

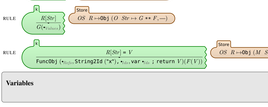
RULE $\frac{}{\mathit{isKResult}(\texttt{return}\ \_;)\ =\ \mathit{true}}$

SYNTAX $Expr ::= FuncObj$

CONFIGURATION:

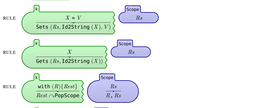
RULE $\frac{V\ \downarrow\ St}{St}$

Object Creation

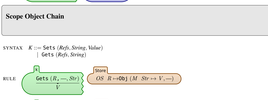
RULE


RULE


RULE


RULE


RULE


RULE


RULE $\frac{\bullet_{\mathit{notAFuncObj}} \curvearrowright R}{R}$

Field Access and Mutation

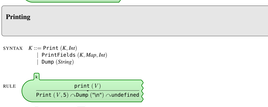
RULE


RULE


RULE

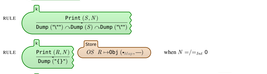

RULE

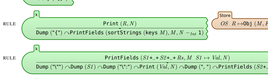

RULE


RULE


Variables

SYNTAX $K ::= \texttt{PopScope}$

RULE


RULE


[structural]

RULE


RULE


RULE


RULE


Scope Object Chain

SYNTAX $K ::= \texttt{Sets}\ (Refs, String, Value)$
| $\texttt{Gets}\ (Refs, String)$

RULE


RULE

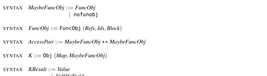

RULE


RULE


RULE


Functions

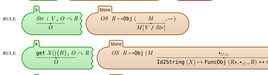
SYNTAX $K ::= \texttt{FEnd}\ (Rs\ell)$

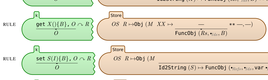
RULE


RULE


RULE


RULE

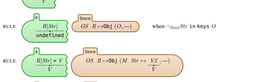
RULE


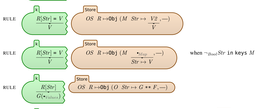
RULE


RULE


Printing

SYNTAX $K ::= \texttt{Print}\ (K, Int)$
| $\texttt{PrintFields}\ (K, Map, Int)$
| $\texttt{Dump}\ (String)$

RULE


RULE


RULE

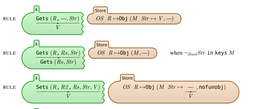
RULE

RULE

RULE

RULE

RULE

RULE


RULE

when $N == _{Int}\ 0$

Sorting

SYNTAX $Strings ::= List\{String, "+\ \_\ +"\}$
| $\texttt{sortStrings}\ (Set)$
| $\texttt{insertString}\ (String, Strings)$

RULE $\frac{}{\texttt{sortStrings}\ (S1\ \ S)\ =\ \texttt{insertString}\ (S1, \texttt{sortStrings}\ (R))}$ [anywhere]

RULE $\frac{}{\texttt{sortStrings}\ (\bullet_{Set}) = \bullet_{Strings}}$ [anywhere]

RULE $\frac{}{\texttt{insertString}\ (S1, \bullet_{Strings}) = S1 \ast \bullet_{Strings}}$ [anywhere]

RULE $\frac{S2 <_{String} S1}{\texttt{insertString}\ (S1, S2 \ast \bullet Rs) = S2 \ast \texttt{insertString}\ (S1, Rs)}$ when $S2$ <String $S1$ [anywhere]

RULE $\frac{S2 \leq_{String} S1}{\texttt{insertString}\ (S1, S2 \ast \bullet Rs) = S2 \ast \texttt{insertString}\ (S1, Rs)}$ when $S2$ ≤String $S1$ [anywhere]

END MODULE