Formal, Executable Semantics of Web Languages: JavaScript and PHP

Sergio Maffeis
Imperial College London

In collaboration with:
J. Mitchell (Stanford), A. Taly (Google),
K. Bhargavan, M. Bodin, A. Charugeraud, A. Delignat-Lavaud, A. Schmitt (INRIA),
D. Filaretti, P. Gardner, D. Naudziuniene, G. Smith, S. Yuwen (Imperial)

PiP'14, San Diego

A Personal Perspective

- Goal: *"language based web security"*
  - 1st step: build formal models (this talk)
  - Next, analyze security properties
- Based on:
  - JSSec: small-step operational semantics of ES3
  - JSCert: Coq semantics and interpreter of ES5
  - KPHP: formal executable semantics of PHP in K
- (Not a literature survey, see my papers for references)

: Principles in Practice

- Given a language L and an interpreter X, define a semantics S such that for all p in L, S(p) ~=~ X(p)
- Real world: here's an interpreter X. Good luck!
  - Define a semantics S such that S(p) === X(p) for as many p as possible
- Approach
  - "Observe" a piece of syntax (experiments & documentation)
  - Model behaviour using building blocks of meta-language
  - Formulate predictions to validate model (testing)

Handling Pre-Existing Systems Complexity

JavaScript and PHP

- Born as small languages
  - JavaScript: sanitize input of HTML forms
  - PHP: Personal Home Page Tools for tracking home page visits
- Now achieved world domination
  - All web pages, most servers
  - Top of Github/StackOveflow popularity
    - Chart from http://langpop.corger.nl
- Picked up lots of complexity along the way

JavaScript and PHP

- Critical points of failure for web security
  - Attacks come from obscure, difficult corner cases
  - Do not leave out tricky or inelegant constructs

```
<a href="#" onclick="b()"> Test B (Safari, Opera and Chrome)</a>
<script>
function b(){
        try {throw (function(){return this});}
        catch (get_scope){get_scope().ref=function(x){return x};
        this.alert("Hacked!")}}
</script>
```

- OK to look at **conservative subsets**
  - But beware of unsound simplifications

```
$arr = array("one", "two", "three");
foreach ($arr as $value) {
    echo "Value: $value<br />\n";
}
```

$\simeq$

```
$arr = array("one", "two", "three");
reset($arr);
while (list(, $value) = each($arr)) {
    echo "Value: $value<br />\n";
}
```

JavaScript and PHP

- Critical points of failure for web security
  - Attacks come from obscure, difficult corner cases
  - Do not leave out tricky or inelegant constructs

```
<a href="#" onclick="b()"> Test B (Safari, Opera and Chrome)</a>
<script>
function b(){
            try {throw (function(){return this});}
            catch (get_scope){get_scope().ref=function(x){return x};
            this.alert("Hacked!")}}
</script>
```

- OK to look at **conservative subsets**
  - But beware of **unsound** simplifications

```
$arr = array("one", "two", "three");
foreach ($arr as $value) {
    echo "Value: $value<br />\n";
}
```

$\neq$

```
$arr = array("one", "two", "three");
reset($arr);
while (list(, $value) = each($arr)) {
    echo "Value: $value<br />\n";
}
```

Libraries

- JavaScript, PHP = Master
- Browser, server = Blaster
- We need *operational* semantics of the core language
  - Plus a mechanism to invoke library functions
- Formalization of libraries is an independent task
  - Different goals, techniques
  - One language, many libraries

Developing and Using Semantics at Scale

Formalization: The Pain

```
// Evaluate the first argument to foreach (the array or object to be iterated)

context 'ForEach(HOLE,, _:K,, _:K)

// if a reference is obtained, read the corresponding location

rule [foreach-arg2Loc]:
        <k> 'ForEach((R:ConvertibleToLoc => convertToLoc(R,r)),,_:K,,_:K) ... </k>
        <trace> Trace:List => Trace ListItem("foreach-arg2Loc") </trace>
        [intermediate]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        write(V,Lx) ~>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L,Lx))) ~>
                        foreach(Lx, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Array,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N <=Int 1) andBool (fresh(Lx:Loc)))
        [step]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        write(V,Lx) ~>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L, none))) ~>
                        foreach(Lx, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Object,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N <=Int 1) andBool (fresh(Lx:Loc)))
        [step]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L, none))) ~>
                        foreach(L, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Value,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N >Int 1))
        [step]

// Error cases: invalid argument

rule [foreach-scalar-1]:
        <k> 'ForEach(L:Loc,,_) =>
                WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <heap> ... L |-> zval(V:Value,_,_,_) ... </heap>
        <trace> Trace:List => Trace ListItem("foreach-scalar-1") </trace>
        when notBool (V isCompoundValue)
        [step, error]

rule [foreach-scalar-2]:
        <k> 'ForEach(V:ScalarValue,,'Pattern(_),,Stmt:K) =>
                WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-scalar-2") </trace>
        [step, error]

rule [foreach-locNull]:
        <k> 'ForEach(Arg:K,,'Pattern(_),,Stmt:K) =>
        WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-locNull") </trace>
        when (Arg ==K locNull)
        [step, error]

// Invalid pattern

rule [foreach-invalid-pattern]:
        <k> 'ForEach(_,, 'Pattern('Some('Key(K:K)),,_),,_) => ERROR("Key element cannot be a reference in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-invalid-pattern") </trace>
        when getKLabel(K) ==KLabel 'Ref
        [step, error]
```

```
$a = array('a', 'b', 'c');
foreach ($a as &$v) {}; // aliasing
foreach ($a as $v) {};
```

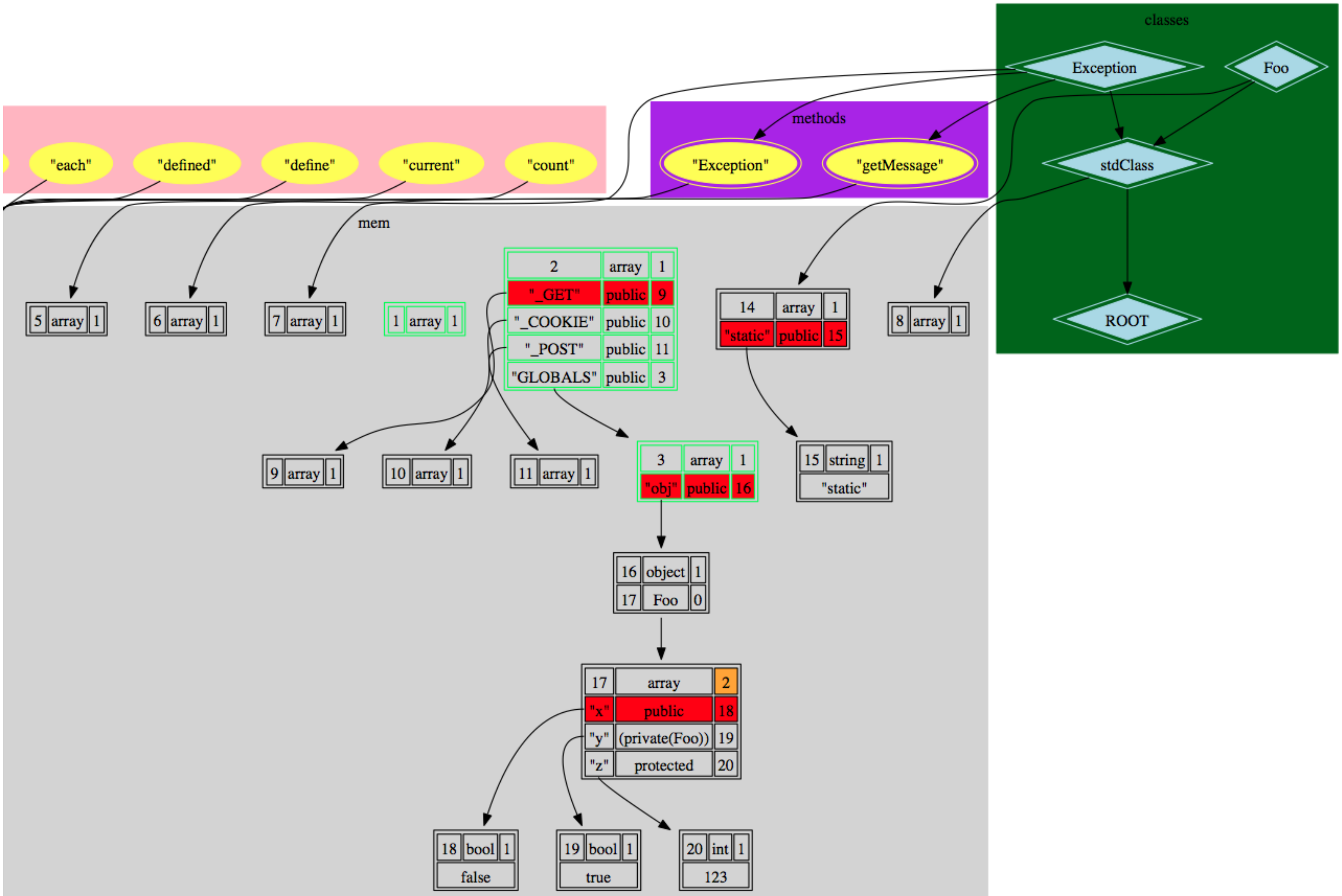
Formalization: The Pain

```
// Evaluate the first argument to foreach (the array or object to be iterated)

context 'ForEach(HOLE,, _:K,, _:K)

// if a reference is obtained, read the corresponding location

rule [foreach-arg2Loc]:
        <k> 'ForEach((R:ConvertibleToLoc => convertToLoc(R,r)),,_:K,,_:K) ... </k>
        <trace> Trace:List => Trace ListItem("foreach-arg2Loc") </trace>
        [intermediate]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        write(V,Lx) ~>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L,Lx))) ~>
                        foreach(Lx, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Array,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N <=Int 1) andBool (fresh(Lx:Loc)))
        [step]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        write(V,Lx) ~>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L, none))) ~>
                        foreach(Lx, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Object,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N <=Int 1) andBool (fresh(Lx:Loc)))
        [step]

rule [foreach]:
        <k> ('ForEach(L:Loc,,Pattern:K,,Stmt:K) ~> K:K) =>
                        pushLoopContext(loopFrame(K, foreachArrayPair(L, none))) ~>
                        foreach(L, Pattern, Stmt) ~>
                        popLoopContext
        </k>
        <heap> ... L |-> zval(V:Value,_,N,_) ... </heap>
        <currentForeachItem> _ => L </currentForeachItem>
        <trace> Trace:List => Trace ListItem("foreach") </trace>
        when ((V isCompoundValue) andBool (N >Int 1))
        [step]

// Error cases: invalid argument

rule [foreach-scalar-1]:
        <k> 'ForEach(L:Loc,,_) =>
                WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <heap> ... L |-> zval(V:Value,_,_,_) ... </heap>
        <trace> Trace:List => Trace ListItem("foreach-scalar-1") </trace>
        when notBool (V isCompoundValue)
        [step, error]

rule [foreach-scalar-2]:
        <k> 'ForEach(V:ScalarValue,,'Pattern(_),,Stmt:K) =>
                WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-scalar-2") </trace>
        [step, error]

rule [foreach-locNull]:
        <k> 'ForEach(Arg:K,,'Pattern(_),,Stmt:K) =>
        WARNING("Warning: Invalid argument supplied for foreach() in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-locNull") </trace>
        when (Arg ==K locNull)
        [step, error]

// Invalid pattern

rule [foreach-invalid-pattern]:
        <k> 'ForEach(_,, 'Pattern('Some('Key(K:K)),,_),,_) => ERROR("Key element cannot be a reference in %s on line %d\n") ... </k>
        <trace> Trace:List => Trace ListItem("foreach-invalid-pattern") </trace>
        when getKLabel(K) ==KLabel 'Ref
        [step, error]
```

```php
$a = array('a', 'b', 'c');
foreach ($a as &$v) {}; // aliasing
foreach ($a as $v) {};
```

```
array(3) { [0]=> string(1)  "a"
           [1]=> string(1)  "b"
           [2]=> string(1)  "b" }
```

Mechanization: The Gain



Parsing

- Manual or lightweight parsing
  - Ok for small projects, not scalable
- A “user-friendly” parser
  - Will get you started quickly but sometimes may be wrong
  - JSCert: based on Closure/Rhino
  - KPHP: based on PHP-front
- A “production” parser
  - Tried with Chromium AST: optimizations get in the way
- Parsing should be verified
  - Also source of security problems (XSS,SQLI,...)

Execution and Testing

- JSSec: manual execution (not scalable)
  - Experiments with various browsers
  - Driven by corner cases of specification
- JSCert: Coq to OCAML extraction
  - JSRef + proof: significant overhead, but **trusted**
  - Systematic validation of JSRef using test262
- KPHP: semantics is directly executable
  - PHP has no analogous to ES3/5 specification
  - (Zend) **test-driven *semantics* development**

Testing, Proofs and Analyses

Coverage

- Lots of possible criteria (Daniel's talk)
- JSCert: LOC
  - Mapping interpreter code/semantics rules
  - Bisect: general-purpose tool for LOC coverage
  - test262: ~95% LOC
- KPHP: ROS
  - Interpreter as black box
  - Instrumentation of semantics with rule traces
  - Zend tests (56% ROS) + our own tests: 100% ROS
- Open problem: automatically derive conformance test suite from formal semantics

Meta-Proofs

- JSSec: paper proof, labor intensive, error-prone

Theorem 1 (Progress and Preservation). *For all states* $S = (H, l, t)$ *and* $S' = (H', l', t')$*:*

- $(Wf(S) \wedge S \to S') \Rightarrow Wf(S')$ *(Preservation)*
- $Wf(S) \wedge t \notin v(t) \Rightarrow \exists S' (S \to S')$ *(Progress)*

where $v(t) = ve$ *if* $t \in Expr$ *and* $v(t) = co$ *if* $t \in Stmnt$ *or* $Prog$.

- JSCert: Coq proof, even more labor, but **trusted**

```
Theorem run_javascript_correct : ∀(n:nat) (p:prog) (o:out),
  run_javascript (runs n) p = result_some (specret_out o) →
  red_javascript p o.
```

- Useful for debugging the semantics
- Basis for further proofs
  - Coq proof: 6 months to find the right way, 3 days to do

Analyses

- Secure subsets, Defensive JavaScript, Program logics
  - Proofs of **reduction-closed invariants** need only semantic rules used by subset
- Temporal verification of PHP programs
  - Based on built-in symbolic execution and LTL model checking
  - Verification tools based on meta-language **cover whole semantics**
- PHP taint analysis based on abstract interpretation
  - Easy to **turn executable semantics into static analyzer**

Engaging With the Industrial Communities

Language Evolution

- JSSec: formalizes ES3
- Horwat: Lisp interpreter for JavaScript 2.0/ES4
- Herman & Flanagan: ES4 specification in ML
- Lambda-JS: ES3 and now ES5S
- JSCert: starts with ES5, open ended
- Language evolution is indeed a challenge
  - Not a good excuse to avoid formalizations
  - You can design a semantics with evolution in mind

DESIGN FOR EVOLUTION: ES5 - JSCERT

12.6.2 The `while` Statement

The production *IterationStatement* **: `while` (** *Expression* **)** *Statement* is evaluated as follows:

1. Let *V* = empty.
2. Repeat
   a. Let *exprRef* be the result of evaluating *Expression*.
   b. If ToBoolean(GetValue(*exprRef*)) is **false**, return (normal, *V*, empty).
   c. Let *stmt* be the result of evaluating *Statement*.
   d. If *stmt*.value is not empty, let *V* = *stmt*.value.
   e. If *stmt*.type is not continue || *stmt*.target is not in the current label set, then
      i. If *stmt*.type is break and *stmt*.target is in the current label set, then
         1. Return (normal, *V*, empty).
      ii. If *stmt* is an abrupt completion, return *stmt*.

```
| red_stat_while : forall S C labs e1 t2 o,
    red_stat S C (stat_while_1 labs e1 t2 resvalue_empty) o ->
    red_stat S C (stat_while labs e1 t2) o

| red_stat_while_1 : forall S C labs e1 t2 rv y1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) y1 ->
    red_stat S C (stat_while_2 labs e1 t2 rv y1) o ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o

| red_stat_while_2_false : forall S0 S C labs e1 t2 rv,
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S false)) (out_ter S rv)

| red_stat_while_2_true : forall S0 S C labs e1 t2 rv o1 o,
    red_stat S C t2 o1 ->
    red_stat S C (stat_while_3 labs e1 t2 rv o1) o ->
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S true)) o

| red_stat_while_3 : forall rv S0 S C labs e1 t2 rv' R o,
    rv' = (If res_value R <> resvalue_empty then res_value R else rv) ->
    red_stat S C (stat_while_4 labs e1 t2 rv' R) o ->
    red_stat S0 C (stat_while_3 labs e1 t2 rv (out_ter S R)) o

| red_stat_while_4_continue : forall S C labs e1 t2 rv R o,
    res_type R = restype_continue /\ res_label_in R labs ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_4_not_continue : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_continue /\ res_label_in R labs) ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_5_break : forall S C labs e1 t2 rv R,
    res_type R = restype_break /\ res_label_in R labs ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) (out_ter S rv)

| red_stat_while_5_not_break : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_break /\ res_label_in R labs) ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o

| red_stat_while_6_abort : forall S C labs e1 t2 rv R,
    res_type R <> restype_normal ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) (out_ter S R)

| red_stat_while_6_normal : forall S C labs e1 t2 rv R o,
    res_type R = restype_normal ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o


| red_stat_abort : forall S C extt o,
    out_of_ext_stat extt = Some o ->
    abort o ->
    ~ abort_intercepted_stat extt ->
    red_stat S C extt o
```

Reporting Bugs

- JSSec:
  - Implementation inconsistencies in browsers
  - (Security) bugs in FBJS, ADSafe, etc.
- JSCert:
  - Bugs in SpiderMonkey, V8, WebKit
  - Problems with ES6, test262
- KPHP:
  - Several horror stories (= bugs)
  - No PHP spec: "It's not a bug! It's a feature!!"

PHP: What is a Bug?

- Evaluation order of expressions: LR or RL?

```
$a = array("one");
$c = $a[0].($a[0] = "two");
echo $c; // prints "onetwo"
```

```
$a = "one";
$c = $a.($a = "two");
echo $c; // prints "twotwo"
```

- PHP bug 61188

[2012-02-26 19:04 UTC] rasmus@php.net

```
I do see your argument, but you are making assumptions about how PHP handles
sequence points in expressions which is not documented and thus not stricly
defined.
```

[2012-09-01 19:01 UTC] avp200681 at gmail dot com

```
[...]
I've found in PHP documentation:
"Operators on the same line have equal precedence, in which
case associativity decides the order of evaluation."
```

PHP: What is a Bug?

- Formal semantics explains what happens

```
$a = array("one");
$c = $a[0].($a[0] = "two");
echo $c; // prints "onetwo"
```

```
$a = "one";
$c = $a.($a = "two");
echo $c; // prints "twotwo"
```

  - Evaluation order **is** LR
  - Array accesses are evaluated to values
  - Variables are evaluated to references
  - References are resolved lazily
- Easy fix to expose LR evaluation consistently
  - BinOp(**E1**,E2) ➔ BinOp(**R**, E2) ➔ BinOp(V,**E2**)

Conclusions

- Toy models of programming languages
  - Ok for new language features, analysis ideas.
  - Inadequate to provide security guarantees
- Full-blown formal semantics
  - Basis for trustworthy verification, certification.
  - Tools and techniques are now mature enough.

References

- JSSec:
  - Semantics: APLAS'08, http://jssec.net/semantics
  - Secure subsets: CSF'09, ESORICS'09, OAKLAND'10
  - Program logics: POPL'12
  - Defensive JavaScript: USENIX'13, http://defensivejs.com
- JSCert:
  - POPL'14 http://jscert.org, https://github.com/jscert/jscert
- KPHP:
  - Submitted. TR available 12/2/14 on http://www.doc.ic.ac.uk/~maffeis/