Dependent Types for JavaScript

Ravi Chugh    Ranjit Jhala
University of California, San Diego

David Herman
Mozilla Research
Goal: Precise and Flexible Reasoning for Fine-Grained Security

But hard even for simple type invariants!
Outline

Challenges

Tour of DJS

Security Predicates
var readLinks = function (doc, max) {

if (!max) max = 10

if (doc.domain() == "newyorker.com") {
    var elts = doc.getEltsByTagName("a")
    for (var i = 0; i < elts.length && i <= max; i++) {
        elts[i].getAttr("href")
    }
}
}

readLinks(document, 5) // read at most 5 links ...
readLinks(document) // ... or 10 by default

Challenges: Unions and Mutation

integer or undefined…
Challenges: Unions and Mutation

```javascript
var readLinks = function (doc, max) {
    if (!max) max = 10
    if (doc.domain() == "newyorker.com") {
        var elts = doc.getEltsByTagName("a")
        for (var i = 0; i < elts.length && i <= max; i++) {
            elts[i].getAttr("href")
        }
    }
}
readLinks(document, 5)
readLinks(document)
```

Challenges: Unions and Mutation
Challenges: Unions and Mutation

```javascript
var readLinks = function (doc, max) {
    if (!max) max = 10
    for (var i = 0; i <= max; i++) {
        elts[i].getAttr("href")
    }
}
readLinks(document, 5)
readLinks(document)
```

integer or undefined...

... but now definitely an integer

i <= max
Challenge: Objects

```javascript
var readLinks = function (doc, max) {
    if (!max) max = 10
    if (doc.domain() == "newyorker.com") {
        var elts = doc.getEltsByTagName("a")
        for (var i = 0; i < elts.length && i <= max; i++) {
            elts[i].getAttr("href")
        }
    }
}
readLinks(document, 5)
readLinks(document)
```
Challenge: Arrays

```javascript
var readLinks = function (doc, max) {
    if (!max) max = 10
    if (doc.domain() == "newyorker.com") {
        var elts = doc.getEltsByTagName("a")
        for (var i = 0; i < elts.length && i <= max; i++) {
            elts[i].getAttr("href")
        }
    } else {
        readLinks(document, 5)
    }
}
readLinks(document)
```

“length”, “holes”, non-integer keys, prototypes
Dependent types

Coq

refinement types

+ nested refinements

occurrence types

F ≤

∨, ∧

syntactic types

dependent types

System D [POPL ’12]

Dependent JavaScript [OOPSLA ’12]

DJS

"Usability"

Expressivity

Expression
Outline

Challenges

Tour of DJS

Security Predicates
| Refinements | Path and Flow Sensitivity | Arrays | Loops | Prototypes |
|-------------|---------------------------|--------|-------|------------|

Refinement Types

\{ x \mid p \}

“value \( x \) such that formula \( p \) is true”

\begin{align*}
\text{Bool} & \equiv \{ b \mid \text{tag}(b) = \text{“boolean”} \} \\
\text{Num} & \equiv \{ n \mid \text{tag}(n) = \text{“number”} \} \\
\text{Int} & \equiv \{ i \mid \text{tag}(i) = \text{“number”} \land \text{integer}(i) \} \\
\text{Top} & \equiv \{ x \mid \text{true} \}
\end{align*}
Refinement Types

\{ x \mid p \} 

“value X such that formula p is true”

3 :: Num

3 :: Int

3 :: \{ i \mid i > 0 \}

3 :: \{ i \mid i = 3 \}
Subtyping is Implication

\{ i \mid i = 3 \} \llt ; \llt \{ i \mid i > 0 \} \llt ; \text{Int} \llt ; \text{Num}

\begin{align*}
i &= 3 \\
\Rightarrow i &> 0 \\
\Rightarrow \text{tag}(i) &= \text{“number”} \land \text{integer}(i) \\
\Rightarrow \text{tag}(i) &= \text{“number”}
\end{align*}
Nested Refinements

McCarthy’s decidable theory of arrays

\{ d \mid \text{Bool}(\text{sel}(d, "f")) \land \text{sel}(d,k) :: \text{Int} \to \text{Int} \}\}

uninterpreted System D “has-type” predicate nests typing relation inside formulas
Nested Refinements

Subtyping is Implication*

\[ \text{Implication}^* = \text{Uninterpreted Validity} + \text{Syntactic Subtyping} \]
Path and Flow Sensitivity

/*: readLinks :: (Ref(~doc), Int?) → Top */

var readLinks = function (doc, max) {
  if (!max) {
    max = 10
  } else {
    (max0 ≠ undefined)
  }
}

T? ≡ \{ x \mid T(x) \lor x = \text{undefined} \}
Path and Flow Sensitivity

/*: readLinks :: (Ref(~doc), Int?) → Top */

var readLinks = function (doc, max) {
    if (!max) {
        max = 10
    } else {
    }
    ...

T? ≡ \{ x | T(x) \vee x = \text{undefined} \}
Flow Sensitivity

\[ \text{var } x = \{ \} \]
\[ x[k] = 7 \]
\[ x_0: \text{Empty} \]
\[ x_1: \{ v | v = \text{upd}(x_0, k, 7) \} \]

Strong updates to singleton objects
Weak updates to collections of objects
Track types, “packedness,” and length of arrays where possible

\[
\{ \ a \ | \ a :: \text{Arr}(A) \ \} \quad \wedge \text{packed}(a) \quad \wedge \text{len}(a) = 10 \}
\]

\[
\{ x | A(x) \lor x = \text{undefined} \}
\]

\[
A? \equiv \{ x | A(x) \lor x = \text{undefined} \}
\]

\[
X \equiv \{ x | x = \text{undefined} \}
\]
(Quick Detour)

Function types include local heap pre- and post-conditions à la separation logic

\[ x : T_1/H_1 \rightarrow T_2/H_2 \]

- input type
- input heap
- output type
- output heap
\[
\text{extern} \quad \text{getIdx} :: \text{All} \quad A. \\
(a: \text{Ref}, \ i: \text{Int}) \\
\text{/} (a_0: \text{Arr}(A)) \\
\rightarrow \{ a_1 | (a_1 :: A \lor a_1 = \text{undefined}) \land \\
(\text{packed}(a_0) \Rightarrow \text{ite} (0 \leq i < \text{len}(a_0)) \\
(a_1 :: A)) \\
(a_1 = \text{undefined})\} \\
\text{/} \text{same} \\
(a_1: \{v | v=a_0\}) \\
\text{ite } p \ q_1 \ q_2 = (p \Rightarrow q_1) \land (p \Rightarrow q_2)
\]
extern setIdx :: All A.
(a:Ref, i:Int, y:A)
/ (a₀:Arr(A))
→ Top
/ (a₁ :: a₁ :: Arr(A) ∧
  (packed(a₀) ∧ 0 <= i < len(a₀) ⇒ packed(a₁) ∧ len(a₁) = len(a₀)) ∧
  (packed(a₀) ∧ i = len(a₀) ⇒ packed(a₁) ∧ len(a₁) = len(a₀) + 1))}
| Refinements | Path and Flow Sensitivity | Arrays | Loops | Prototypes |
|-------------|---------------------------|--------|-------|------------|

```c
extern __ArrayProto :: {v | sel(v, "pop") :: ... ∧ sel(v, "push") :: ... ∧ ...
```
var readLinks = function (doc, max) {
    if (!max) max = 10
    if (doc.domain() == "newyorker.com") {
        var elts = doc.getEltsByTagName("a")
        for (var i = 0; i < elts.length && i <= max; i++) {
            elts[i].getAttr("href")
        }
    }
}

---

Refinements

- Path and Flow Sensitivity
- Arrays
- Loops
- Prototypes

---

Heap invariants before and after each iteration

Type checker infers heap for common cases
DJS handles prototypes...
Desugared Program

Based on Guha et al. [ECOOP ’10]

13 benchmarks mainly from SunSpider and JSGP

300 unannotated LOC
35% annotation overhead

11 benchmarks run in <1s
2 benchmarks run in 2-6s
Outline

Challenges

Tour of DJS

Security Predicates
Can we track fine-grained policies directly in JS?

Granularity of Invariants

coarse fine

ADsafety
Politz et al.
[SEC ’11]

DJS

IBEX
Guha et al.
[SP ’11]

“Degree of JavaScript”

JS

ML
readLinks :: (Ref(~doc), Int?) → Top

var readLinks = function (doc, max) {
  if (!max) max = 10
  if (doc.domain() == "newyorker.com") {
    var elts = doc.getEltsByTagName("a")
    for (var i = 0; i < elts.length && i <= max; i++) {
      elts[i].getAttr("href")
    }
  }
}

Allow extension to read this attribute?
`/*: assume forall (e d) (eltTagName e “a” ∨ eltInDoc e d ∧ docDomain d “newyorker.com”) ⇒ canReadAttr e “href” */`  

`/*: readLinks :: (Ref(~doc), Int?)→Top */`  

`var readLinks = function (doc, max) {`  

`if (!max) max = 10`  

`if (doc.domain() == “newyorker.com”) {`  

`var elts = doc.getEltsByTagName(“a”)`  

`for (var i = 0; i < elts.length && i <= max; i++) {`  

`elts[i].getAttr(“href”)`  

`}`  

`}`  

`}`

**IBEX-style security policy**

**Type check against IBEX-style DOM API**
extern Elt.prototype.getAttr ::
    (this:Ref(~elt), k:Str)
→ Str
extern Elt.prototype.getAttr ::
    (this:Ref(~elt), {k | Str(k) ∧ canReadAttr this k})
→ {s | Str(s) ∧ attrOfElt this k s}
extern Elt.prototype.getAttr ::
  (this:Ref(~elt), {k | Str(k) ∧ canReadAttr this k})
  → {s | Str(s) ∧ attrOfElt this k s}

extern Doc.prototype.domain ::
  (this:Ref(~doc))
  → Str
extern Elt.prototype.getAttr ::
    (this:Ref(~elt), \{k \mid \text{Str}(k) \land \text{canReadAttr this k}\} 
    → \{s \mid \text{Str}(s) \land \text{attrOfElt this k s}\}

extern Doc.prototype.domain ::
    (this:Ref(~doc))
    → \{s \mid \text{Str}(s) \land \text{docDomain this s}\}
extern Elt.prototype.getAttr ::
  (this:Ref(~elt), {k | Str(k) ∧ canReadAttr this k})
  → {s | Str(s) ∧ attr0fElt this k s}

extern Doc.prototype.domain ::
  (this:Ref(~doc))
  → {s | Str(s) ∧ docDomain this s}

extern Doc.prototype.getEltsByTagName :: ...
/*: assume forall (e d) (eltTagName e “a” ∧ eltInDoc e d ∧ docDomain d “newyorker.com”) ⇒ canReadAttr e “href” */

/*: readLinks :: (Ref(~doc), Int?)→Top */
var readLinks = function (doc, max) {
  if (!max) max = 10
  if (doc.domain() == “newyorker.com”) {
    var elts = doc.getEltsByTagName(“a”)
    for (var i = 0; i < elts.length && i <= max; i++) {
      elts[i].getAttr(“href”)
    }
  }
}
Current Status

9 of 17 IBEX examples ported to DJS

Total running time ~3s

Invariants translate directly (so far)
Conclusion

DJS able to track simple type invariants, and security predicates seem within reach
Thanks!

ravichugh.com/djs

github.com/ravichugh/djs
Extra Slides
Key Membership via Prototype Chain Unrolling

\[ b :: \{ v \mid v = \text{true} \iff \right. \]

\[
(has(child,k) \lor has(parent,k) \lor has(grandpa,k) \lor HeapHas(H,great,k))
\]

```
var grandpa = ..., 
parent = Object.create(grandpa), 
child = Object.create(parent), 
b = k in child,
```
Key Lookup via Prototype Chain Unrolling

```javascript
var grandpa = ..., parent = Object.create(grandpa), child = Object.create(parent), b = k in child, 
x = child[k]

x :: { v |
  if has(child,k) then v = sel(child,k) 
  elif has(parent,k) then v = sel(parent,k)
  elif has(grandpa,k) then v = sel(grandpa,k)
  elif HeapHas(H,great,k) then v = HeapSel(H,great,k))
  else v = undefined }
```
Key Idea

Reduce prototype semantics to decidable theory of arrays via flow-sensitivity and unrolling
Encode **tuples** as arrays

```javascript
var tup = [5, “guten abend!”]
```

\[
\{ a \mid a :: \text{Arr}(\text{Top}) \\
\quad \land \text{packed}(a) \land \text{len}(a) = 2 \\
\quad \land \text{Int}(a[0]) \\
\quad \land \text{Str}(a[1]) \}
\]
var elts = doc.getEltsByTagName("a")
var i = 0

var loop = function loop () {
  if (i < elts.length && i <= max) {
    elts[i].getAttr("href")
    i++
    loop()
  } else {
    undefined
  }
}
function negate(x) {
    x = (typeof x == "number") ? 0 - x : !x
    return x
}

function negate(x) {
    x = (typeof x == "number") ? 0 - x : !x
    return x
}
Function Types and Objects

\[ x : T_1/H_1 \rightarrow T_2/H_2 \]

input type \quad input heap \quad output type \quad output heap

\[
\text{ObjHas}(d,k,H,d') \equiv \text{has}(d,k) \lor \text{HeapHas}(H,d',k)
\]

/*: x:Ref / [x |-\> d:Dict |-\> ^x] 
   \rightarrow \{ v \iff \text{ObjHas}(d,"f",\text{curHeap},^x) \} / \text{sameHeap} */

function hasF(x) {
   return "f" in x
}

x:T_1/H_1 \rightarrow T_2/H_2
**Function Types and Objects**

\[ x : T_1 \rightarrow T_2 \]

\[ x : T_1 / H_1 \rightarrow T_2 / H_2 \]

- **Input type**: \[ T_1 \]
- **Input heap**: \[ H_1 \]
- **Output type**: \[ T_2 \]
- **Output heap**: \[ H_2 \]

\[ \text{ObjSel}(d,k,H,d') = \text{ite has}(d,k) \text{ sel}(d,k) \text{ HeapSel}(H,d',k) \]

/*: x:Ref / [x |-> d:Dict |> ^x]
   \rightarrow \{ v = \text{ObjSel}(d,"f",\text{curHeap},^x) \} / \text{sameHeap} */

**Function readF(x)**

```javascript
function readF(x) {
    return x.f
}
```