Dependent Types for JavaScript

Ravi Chugh   Ranjit Jhala

University of California, San Diego

David Herman

Mozilla Research

A Large Subset of

Types for JavaScript

Goal:

Precise and Flexible Reasoning
for Fine-Grained Security

But hard even for simple type invariants!

Outline

Challenges

Tour of DJS

Security Predicates

Challenges: Unions and Mutation

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







}
readLinks(document,5) // read at most 5 links …
readLinks(document)  // … or 10 by default
```

integer or undefined…

Challenges: Unions and Mutation

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

integer or undefined…

```
}
readLinks(document,5)
readLinks(document)
```

Challenges: Unions and Mutation

```
var readLinks = function (doc, max) {
  if (!max) max = 10
```

integer or undefined…

… but now definitely an integer

```
i <= max
```

```
}
readLinks(document,5)
readLinks(document)
```

Challenge: Objects

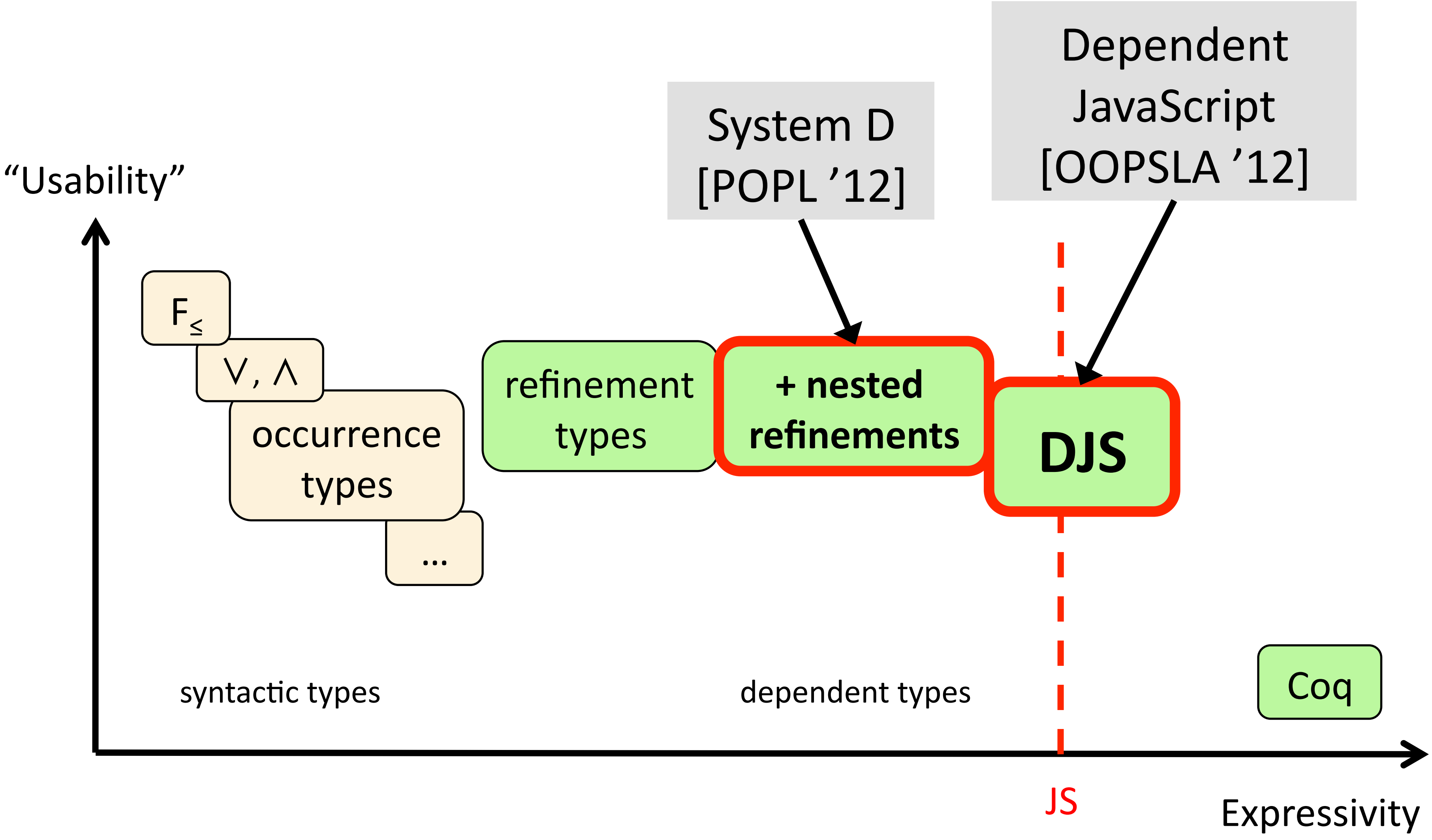
```
var readLinks = function (doc, max) {
  if (!max) max = 10
  if (doc.domain() == “newyorker.com”) {
    var elts = doc.getEltsByTagName(“a”)
                                          i <= max



  }
}
readLinks(document,5)
readLinks(document)
```

prototype inheritance,
mutability, dynamic keys

Challenge: Arrays

```
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)
```

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



System D
[POPL ’12]

Dependent
JavaScript
[OOPSLA ’12]

“Usability”

$F_{\leq}$

$\vee, \wedge$

occurrence types

...

refinement types

+ nested refinements

DJS

syntactic types

dependent types

Coq

JS

Expressivity

Outline

Challenges

Tour of DJS

Security Predicates

Refinement Types

$$\{\, x \mid p \,\}$$

“value x such that formula p is true”

$$\mathtt{Bool} \equiv \{\, \mathtt{b} \mid \mathtt{tag}(\mathtt{b}) = \text{“}\mathtt{boolean}\text{”} \,\}$$

$$\mathtt{Num} \equiv \{\, \mathtt{n} \mid \mathtt{tag}(\mathtt{n}) = \text{“}\mathtt{number}\text{”} \,\}$$

$$\mathtt{Int} \equiv \{\, \mathtt{i} \mid \mathtt{tag}(\mathtt{i}) = \text{“}\mathtt{number}\text{”} \wedge \mathtt{integer}(\mathtt{i}) \,\}$$

$$\mathtt{Top} \equiv \{\, \mathtt{x} \mid \mathtt{true} \,\}$$

Refinement Types

$$\{\, x \mid p \,\}$$

"value x such that formula p is true"

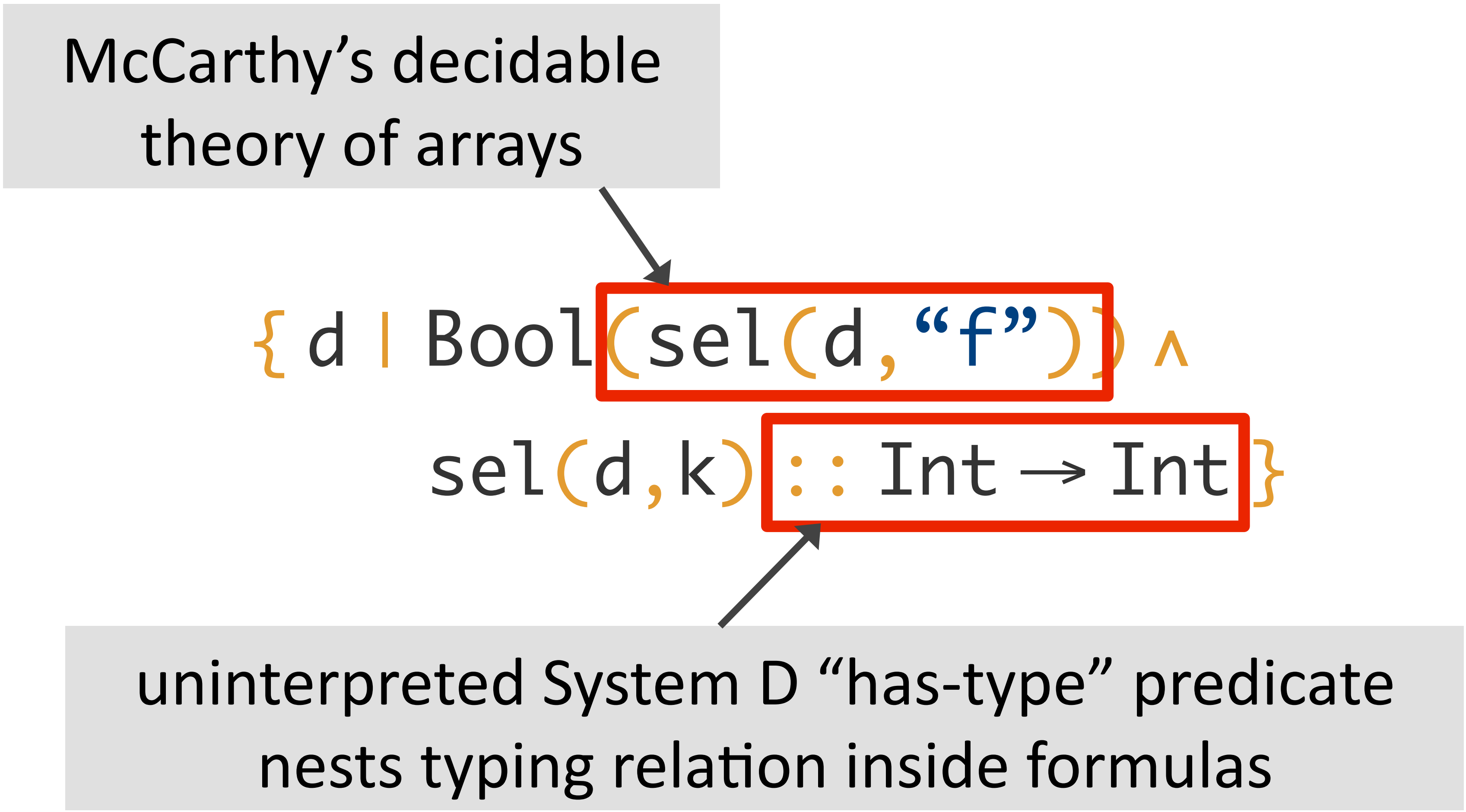
```
3 :: Num

3 :: Int

3 :: {i|i>0}

3 :: {i|i=3}
```

Subtyping is Implication

$$\{\texttt{i} \mid \texttt{i} = 3\} \;<:\; \{\texttt{i} \mid \texttt{i} > 0\} \;<:\; \texttt{Int} \;<:\; \texttt{Num}$$

$$\begin{aligned}
& \texttt{i} = 3 \\
\Rightarrow\; & \texttt{i} > 0 \\
\Rightarrow\; & \texttt{tag}(\texttt{i}) = \text{“}\texttt{number}\text{”} \wedge \texttt{integer}(\texttt{i}) \\
\Rightarrow\; & \texttt{tag}(\texttt{i}) = \text{“}\texttt{number}\text{”}
\end{aligned}$$

Nested Refinements



McCarthy’s decidable theory of arrays

$$\{\, d \mid \mathtt{Bool}(\boxed{\mathtt{sel}(d,\text{“f”})}) \wedge \mathtt{sel}(d,k) \boxed{::\ \mathtt{Int} \rightarrow \mathtt{Int}} \,\}$$

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

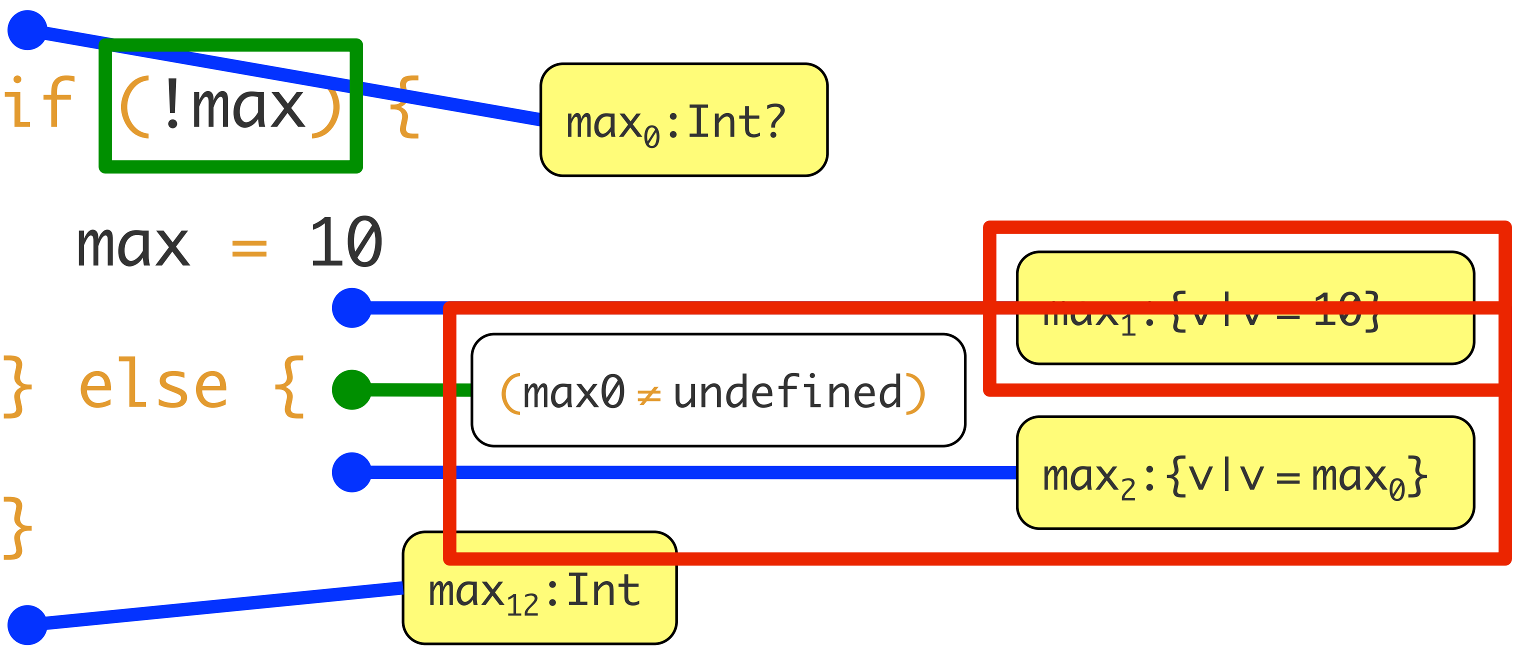
Nested Refinements

Subtyping is Implication*

Implication* = Uninterpreted Validty + Syntactic Subtyping

Path and Flow Sensitivity

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

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



$max_0 : Int?$

$max_1 : \{v \mid v = 10\}$

$(max0 \neq undefined)$

$max_2 : \{v \mid v = max_0\}$

$max_{12} : Int$

$T? \equiv \{ x \mid T(x) \vee x = undefined \}$

Path and Flow Sensitivity

```
/*: readLinks :: (Ref(~doc), Int?) → Top */
var readLinks = function (doc, max) {
```

max_0:Int?

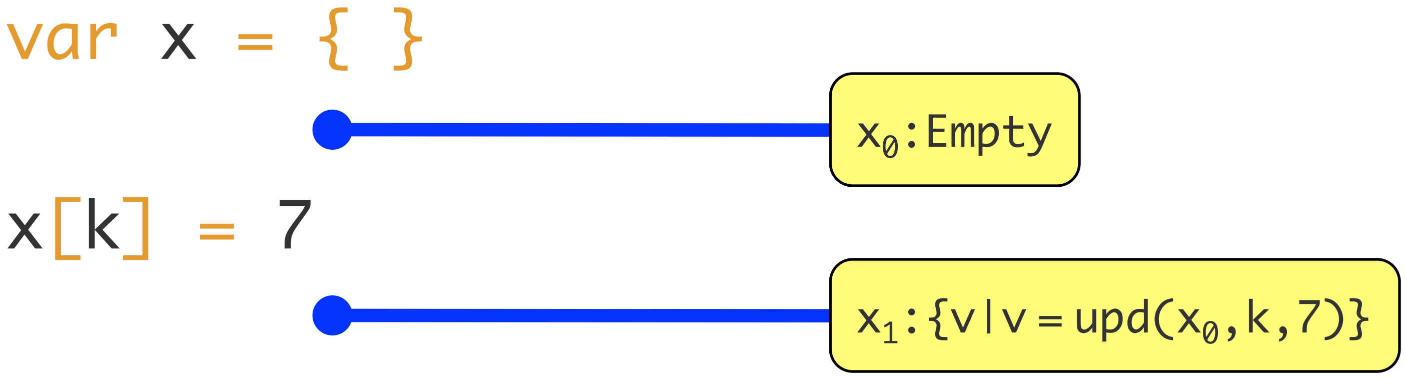
```
  if (!max) {
    max = 10
  } else {
  }
```

max_{12}:Int

```
  …
```

$$T? \equiv \{\, x \mid T(x) \lor x = \text{undefined} \,\}$$

Flow Sensitivity



```
var x = { }
x[k] = 7
```

x_0:Empty

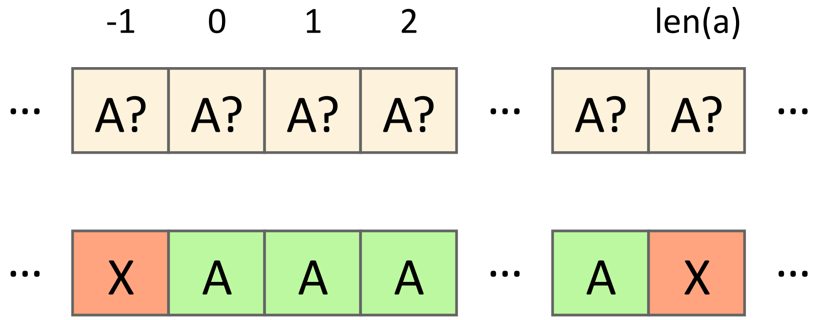
$x_1 : \{v \mid v = \mathrm{upd}(x_0, k, 7)\}$

Strong updates to singleton objects

Weak updates to collections of objects

Track **types**, "**packedness**," and **length** of arrays where possible

$$\{\, a \mid a :: \mathtt{Arr}(A)$$
$$\wedge\ \mathtt{packed}(a)$$
$$\wedge\ \mathtt{len}(a) = 10 \,\}$$



	-1	0	1	2			len(a)	
…	A?	A?	A?	A?	…	A?	A?	…
…	X	A	A	A	…	A	X	…

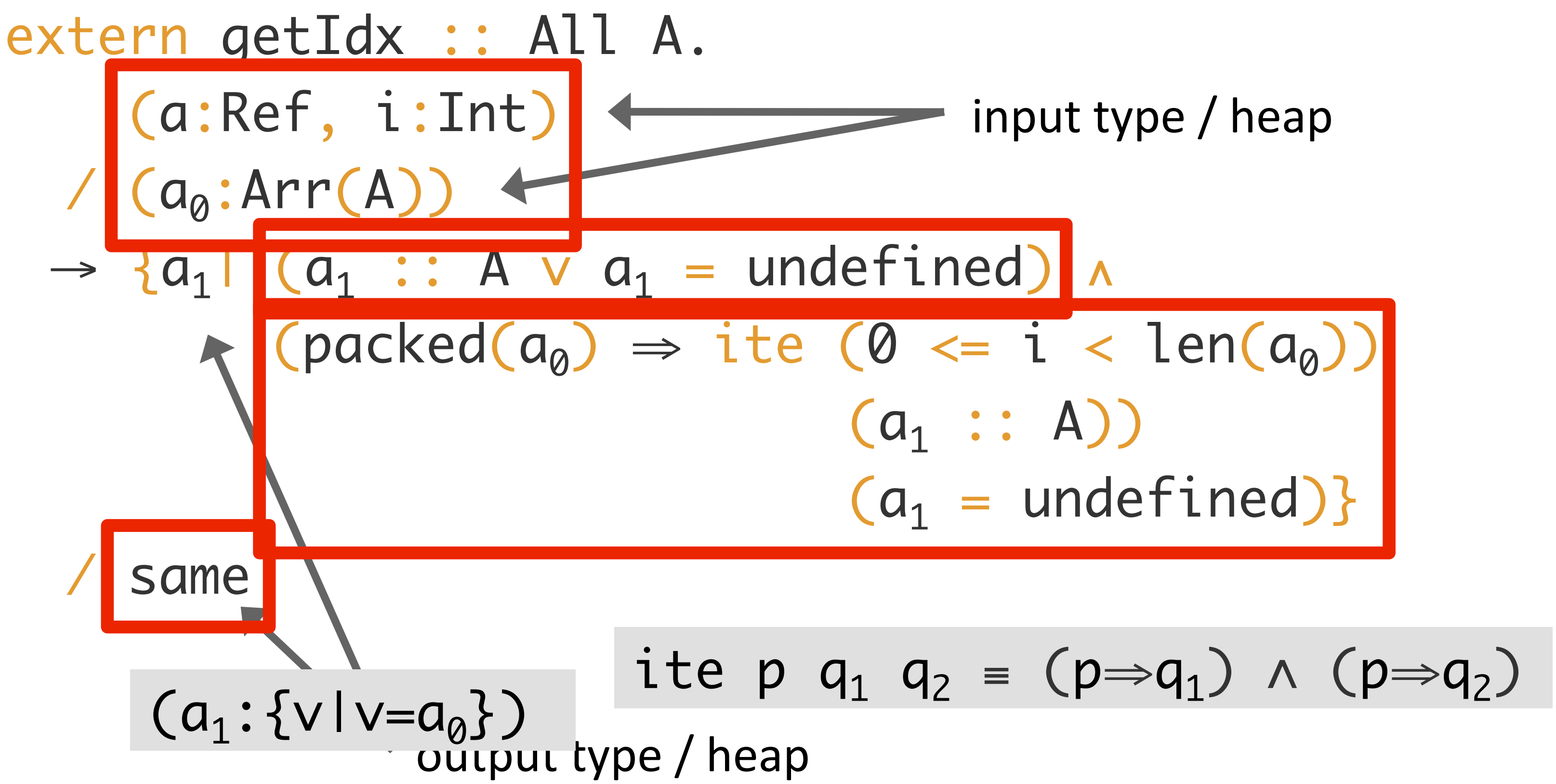
$$A? \equiv \{\, x \mid A(x) \vee x = \mathtt{undefined} \,\}$$

$$X \equiv \{\, x \mid x = \mathtt{undefined} \,\}$$

(Quick Detour)

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

$$x : \underset{\substack{\uparrow \\ \text{input type}}}{T_1} / \underset{\substack{\uparrow \\ \text{input heap}}}{H_1} \rightarrow \underset{\substack{\uparrow \\ \text{output type}}}{T_2} / \underset{\substack{\uparrow \\ \text{output heap}}}{H_2}$$



```
extern getIdx :: All A.
      (a:Ref, i:Int)
    / (a0:Arr(A))
    → {a1| (a1 :: A ∨ a1 = undefined) ∧
           (packed(a0) ⇒ ite (0 <= i < len(a0))
                             (a1 :: A))
                             (a1 = undefined)}
    / same
```

input type / heap

$(a_1:\{v|v=a_0\})$

output type / heap

$\text{ite}\ p\ q_1\ q_2 \equiv (p \Rightarrow q_1) \wedge (p \Rightarrow q_2)$

```
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)})
```

```
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”)
    }
  }
}
```

$\mathtt{max}_{\mathtt{inv}}$: Int

$\mathtt{i}_{\mathtt{inv}}$: $\{v \mid v >= 0\}$

$\mathtt{elts}_{\mathtt{inv}}$: $\{a \mid a = \mathtt{elts}_0\}$

$\mathtt{Elt.proto}_{\mathtt{inv}}$: $\{d \mid \ldots\}$

Heap invariants before and after each iteration

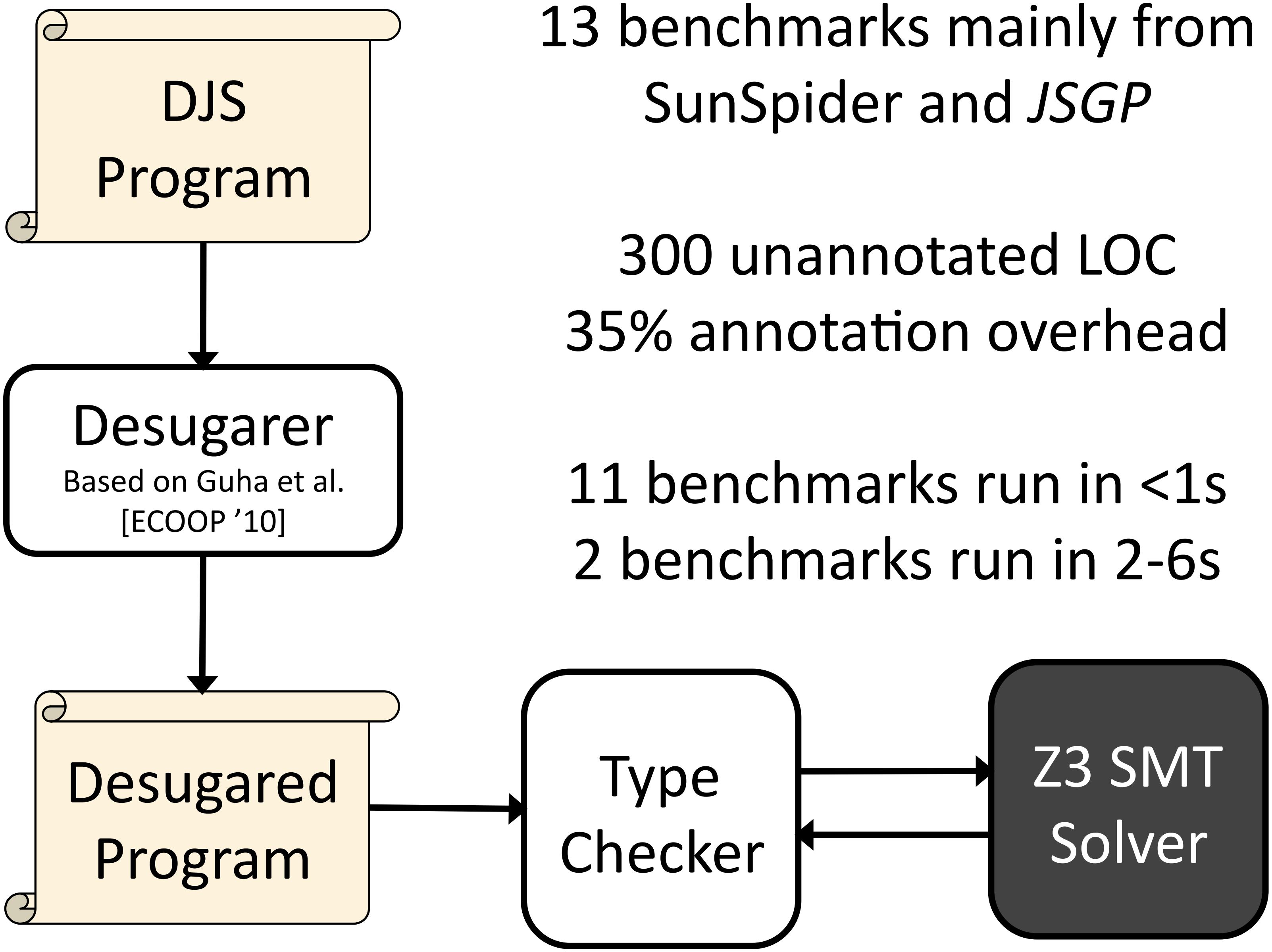
Type checker infers heap for common cases

DJS handles prototypes…

13 benchmarks mainly from SunSpider and *JSGP*

300 unannotated LOC
35% annotation overhead

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



DJS Program

Desugarer
Based on Guha et al. [ECOOP ’10]

Desugared Program

Type Checker

Z3 SMT Solver

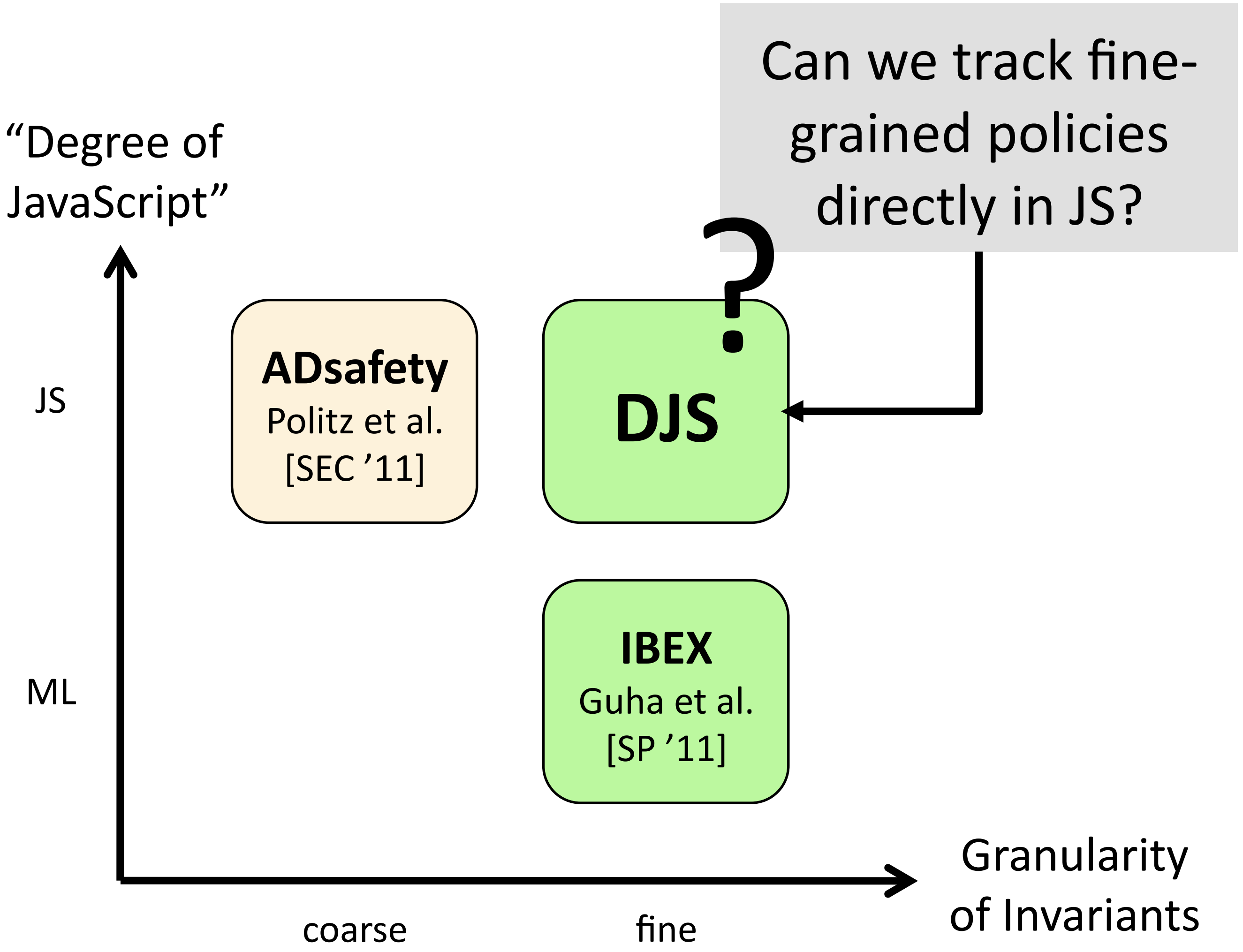
Outline

Challenges

Tour of DJS

Security Predicates

Browser Extension Security



Can we track fine-grained policies directly in JS?

```
/*: 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”          */
```

IBEX-style security policy

```
/*: 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”)
    }
  }
}
```

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 | Str(k) ∧ canReadAttr this k)
 → {s | Str(s) ∧ attrOfElt this k s }


extern Doc.prototype.domain ::
    (this:Ref(~doc))
 → {s | Str(s) ∧ docDomain this 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))
 → {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”          */
```

IBEX-style security policy

```
/*: 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”) ✓
    }
  }
}
```

Type check against IBEX-style DOM API

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

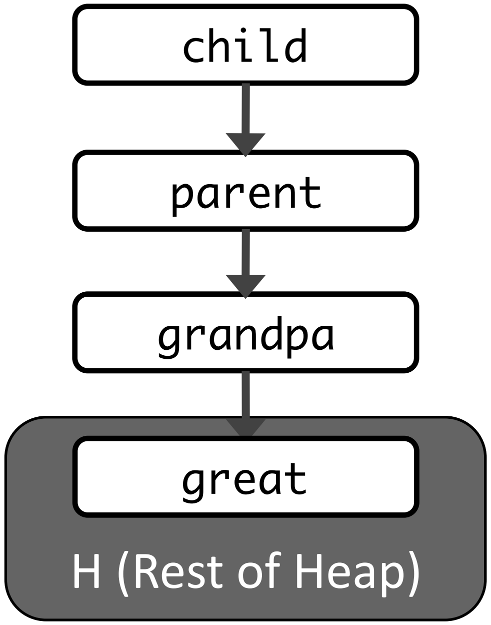
github.com/ravichugh/djs

Extra Slides

Key Membership via Prototype Chain Unrolling

```
var grandpa = …,
    parent  = Object.create(grandpa),
    child   = Object.create(parent),
    b       = k in child,
```

```
b :: {v | v = true iff

          (has(child,k) v

           has(parent,k) v

           has(grandpa,k) v

           HeapHas(H,great,k)) }
```



Key Lookup via Prototype Chain Unrolling

```
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

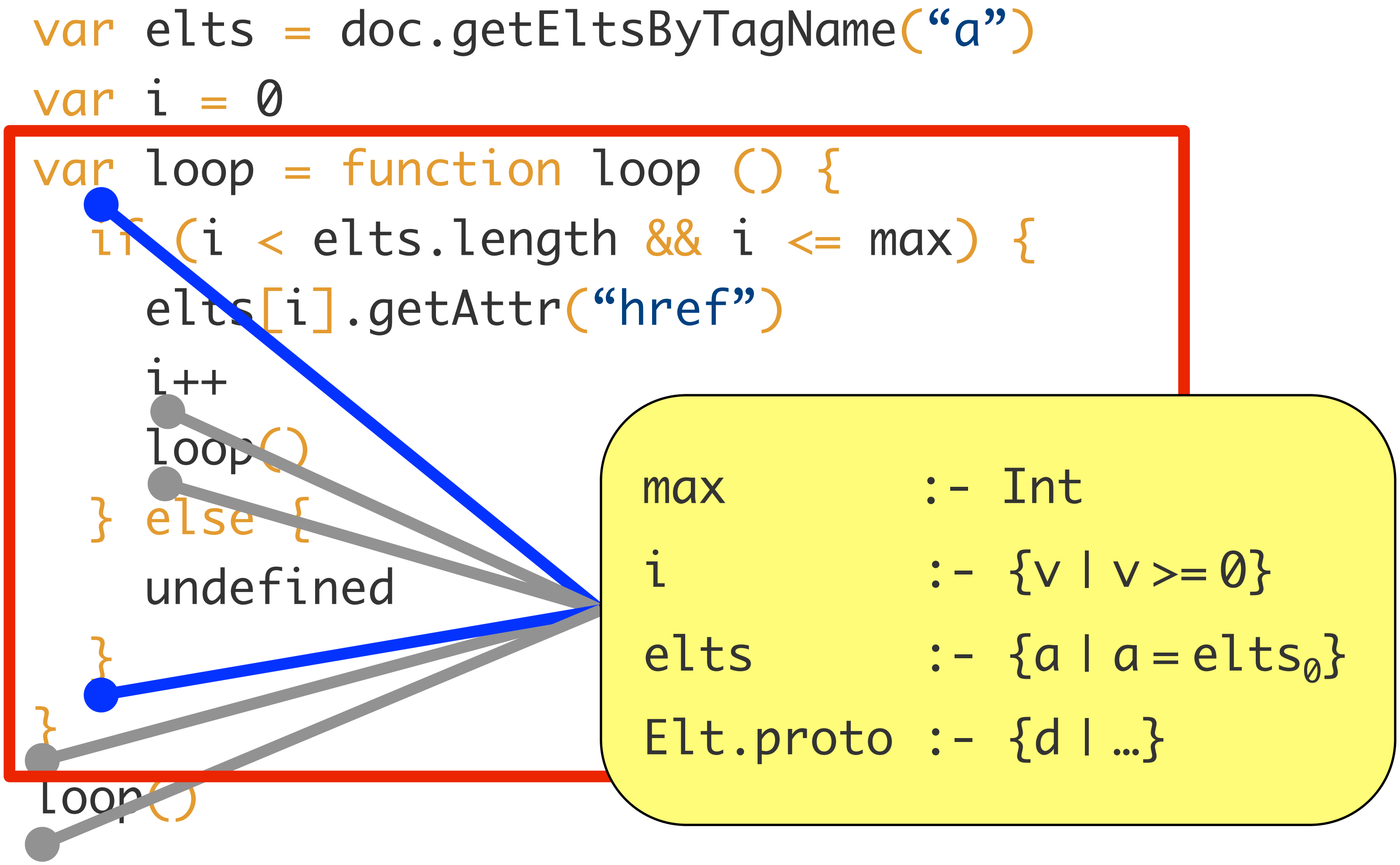
Reduce prototype semantics
to decidable theory of arrays
via flow-sensitivity and unrolling

Encode **tuples** as arrays

```
var tup = [5, “guten abend!”]
```

$$
\{ a \mid a :: \mathtt{Arr}(\mathtt{Top}) \\
\wedge\ \mathtt{packed}(a) \wedge \mathtt{len}(a) = 2 \\
\wedge\ \mathtt{Int}(a[0]) \\
\wedge\ \mathtt{Str}(a[1]) \}
$$

Desugared Loop



```
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
  }
}
loop()
```

max :- Int

i :- $\{v \mid v >= 0\}$

elts :- $\{a \mid a = \mathtt{elts}_0\}$

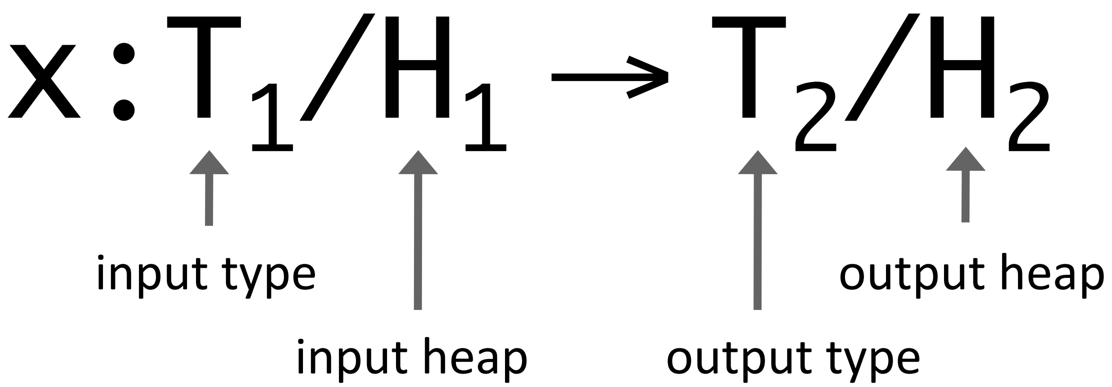
Elt.proto :- {d | …}

{p} ≡ {v|p}

```
/*: x:NumOrBool → {ite Num(x) Num(v) Bool(v)} */
function negate(x) {
  x = (typeof x == “number”) ? 0 – x : !x
  return x
}
```

```
/*: x:Any → {v iff falsy(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

input heap

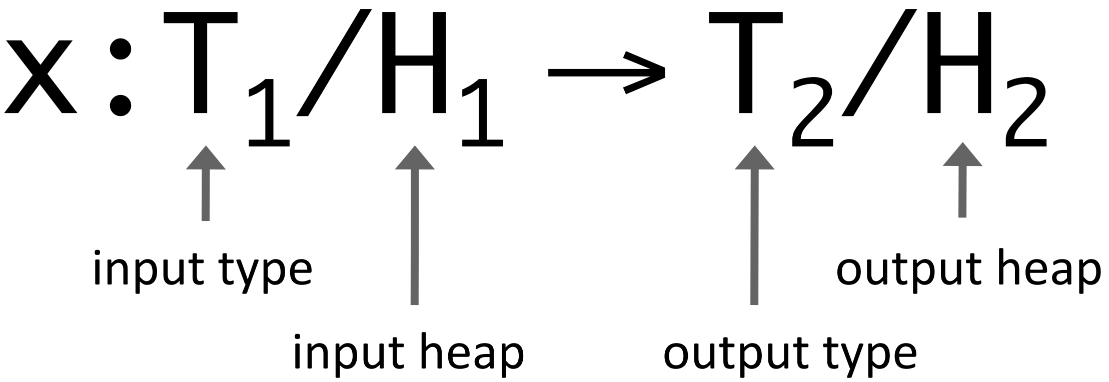
output type

output heap

```
ObjHas(d,k,H,d’) ≡ has(d,k) v HeapHas(H,d’,k)
```

```
/*: x:Ref / [x |-> d:Dict |> ^x]
    → {v iff ObjHas(d,”f”,curHeap,^x)} / sameHeap */
function hasF(x) {
  return “f” in x
}
```

Function Types and Objects



$$x : T_1 / H_1 \rightarrow T_2 / H_2$$

input type

input heap

output type

output heap

```
ObjSel(d,k,H,d’) ≡
        ite has(d,k) sel(d,k) HeapSel(H,d’,k)
```

```
/*: x:Ref / [x |-> d:Dict |> ^x]
    → {v=ObjSel(d,”f”,curHeap,^x)} / sameHeap */
function readF(x) {
  return x.f
}
```