A Light-Weight Effect System for JavaScript

Christopher Schuster, Cormac Flanagan
University of California, Santa Cruz
{cschuste,cormac}@ucsc.edu

The type only tells part of the story

What values does this function accept/compute?

```
inc :: Int -> Int
function inc(i) {

  return i + 1;
}
```

The type only tells part of the story

What values does this function accept/compute?

```
inc :: Int -> Int
function inc(i) {
  alert(“Hi there!”);
  return i + 1;
}
```

Effects of a Function

What effects are involved in computing this function?

```
inc :: Int --(dialog)-->  Int
function inc(i) {
  alert(“Hi there!”);
  return i + 1;
}
```

Example effects: File IO

Input/output, e.g. files...

```
inc :: Int --(fio)-->  Int
function inc(i) {
  fprintf(“file.txt”, “Hi”);
  return i + 1;
}
```

Example effects: Network IO

Input/output, e.g. files, network, ...

```
inc :: Int --(http)-->  Int
function inc(i) {
  post("http://example.org/" + i);
  return i + 1;
}
```

Example effects: Checked exceptions

As known in Java….

```
inc :: Int --(throws[ZeroE])-->  Int
function inc(i) {
  if (i == 0) throw new ZeroE();
  return i + 1;
}
```

Example effects: Changing global state

```
inc :: Int --(modifies[g])-->  Int
function inc(i) {
  global.g++;
  return i + 1;
}
```

A problematic JavaScript effect

“Web workers” crash when accessing the DOM

```
alert(“Hello,”);       // modal dialog
startWorker(function() {
  alert(“world!”);     // crash
});
```

Annotating JavaScript functions

```
function alert() fx[dom] {}
```

- `alert` function with dom effect

```
function startWorker(func fx[!dom]){}
```

- `startWorker` function expecting
  a function argument without dom effects

Demo!

Static analysis

1) Determine whole program control flow with set constraint-based analysis

2) Add effect constraints from annotations and propagate along call graph

3) Unsatisfiable constraint system = effect checking error

Set-constraint based type inference

Enumerate all function definition and object literals

$$\Lambda = \lambda_1, \lambda_2, ...$$

$$\Omega = \{...\}_1, \{...\}_2, ...$$

A “type” is simply a subset of these functions/objects*:

$$\tau \in \mathcal{P}(\Lambda \cup \Omega) \quad \text{(Type approximation)}$$

*We don’t care about primitive types here, because they don’t have effects.

Function effects

Effects are simple user-defined tags (dom,io,network,...)

$$e \in E : \text{Effects}$$

Every function has a set of effects:

$$\phi_f : \Lambda \to \mathcal{P}(E)$$

Set-based analysis: Example

```
alert = fn1() {};

startWorker = fn2(func) {};

startWorker(alert);
```

Set-based analysis: Type constraints

```
alert = fn1() {};

startWorker = fn2(func) {};

startWorker(alert);
```

τ(alert) = {} τ(startWorker) = {} τ(arg1(fn2)) = {}

Set-based analysis: Type constraints

```
alert = fn1() {};                    τ(alert) ⊇ {fn1}

startWorker = fn2(func) {};

startWorker(alert);
```

```
τ(alert) = {}   τ(startWorker) = {}   τ(arg1(fn2)) = {}
```

Set-based analysis: Type constraints

```
alert = fn1() {};
```

$\tau(\text{alert}) \supseteq \{\text{fn}_1\}$

```
startWorker = fn2(func) {};

startWorker(alert);
```

$\tau(\text{alert}) = \{\text{fn}_1\}$ $\tau(\text{startWorker}) = \{\}$ $\tau(\text{arg1}(\text{fn}_2)) = \{\}$

Set-based analysis: Type constraints

```
alert = fn1() {};                 τ(alert) ⊇ {fn1}

startWorker = fn2(func) {};       τ(startWorker) ⊇ {fn2}

startWorker(alert);
```

τ(alert) = {fn1} τ(startWorker) = {} τ(arg1(fn2)) = {}

Set-based analysis: Type constraints

```
alert = fn1() {};                    τ(alert) ⊇ {fn1}

startWorker = fn2(func) {};          τ(startWorker) ⊇ {fn2}

startWorker(alert);
```

τ(alert) = {fn1} τ(startWorker) = {fn2} τ(arg1(fn2)) = {}

Set-based analysis: Type constraints

`alert = fn1() {};`	$\tau(\text{alert}) \supseteq \{fn_1\}$
`startWorker = fn2(func) {};`	$\tau(\text{startWorker}) \supseteq \{fn_2\}$
`startWorker(alert);`	$\forall f \in \tau(\text{startWorker})$ $\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{\}$

Set-based analysis: Type constraints

`alert = fn1() {};`	$\tau(\text{alert}) \supseteq \{fn_1\}$
`startWorker = fn2(func) {};`	$\tau(\text{startWorker}) \supseteq \{fn_2\}$
`startWorker(alert);`	$\forall f \in \tau(\text{startWorker})$ $\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$

Set-based analysis: Effect constraints

`alert = fn₁() {};`	$\tau(\text{alert}) \supseteq \{fn_1\}$
`startWorker = fn₂(func) {};`	$\tau(\text{startWorker}) \supseteq \{fn_2\}$
`startWorker(alert);`	$\forall f \in \tau(\text{startWorker})$ $\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect constraints

```
alert = fn₁() fx[dom] {};
```
$\tau(\text{alert}) \supseteq \{fn_1\}$

```
startWorker = fn₂(func) {};
```
$\tau(\text{startWorker}) \supseteq \{fn_2\}$

```
startWorker(alert);
```
$\forall f \in \tau(\text{startWorker})$
$\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$
$\varphi(fn_1) = \{\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect constraints

```
alert = fn1() fx[dom] {};
```

$\tau(alert) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{dom\}$

```
startWorker = fn2(func) {};
```

$\tau(startWorker) \supseteq \{fn_2\}$

```
startWorker(alert);
```

$\forall f \in \tau(startWorker)$

$\tau(arg1(f)) \supseteq \tau(alert)$

$\tau(alert) = \{fn_1\}$ $\tau(startWorker) = \{fn_2\}$ $\tau(arg1(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect constraints

```
alert = fn₁() fx[dom] {};
```

$\tau(\text{alert}) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{\text{dom}\}$

```
startWorker = fn₂(func) {};
```

$\tau(\text{startWorker}) \supseteq \{fn_2\}$

```
startWorker(alert);
```

$\forall f \in \tau(\text{startWorker})$

$\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{\text{dom}\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect constraints

```
alert = fn₁() fx[dom] {};
```

$\tau(alert) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{dom\}$

```
startWorker =
  fn₂(func fx[!dom]) {};
```

$\tau(startWorker) \supseteq \{fn_2\}$

```
startWorker(alert);
```

$\forall f \in \tau(startWorker)$

$\tau(arg1(f)) \supseteq \tau(alert)$

$\tau(alert) = \{fn_1\}$ $\tau(startWorker) = \{fn_2\}$ $\tau(arg1(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{dom\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect constraints

```
alert = fn₁() fx[dom] {};
```

$\tau(\text{alert}) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{\text{dom}\}$

```
startWorker =
  fn₂(func fx[!dom]) {};
```

$\tau(\text{startWorker}) \supseteq \{fn_2\}$

$\forall f \in \tau(\text{arg1}(fn_2))\ \varphi(f) \nsupseteq \text{dom}$

```
startWorker(alert);
```

$\forall f \in \tau(\text{startWorker})$
$\tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$
$\varphi(fn_1) = \{\text{dom}\}$ $\varphi(fn_2) = \{\}$

Set-based analysis: Effect checking error

```
alert = fn₁() fx[dom] {};
```

$\tau(\text{alert}) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{\text{dom}\}$

```
startWorker =
  fn₂(func fx[!dom]) {};
```

$\tau(\text{startWorker}) \supseteq \{fn_2\}$

$\forall f \in \tau(\text{arg1}(fn_2))\ \varphi(f) \not\ni \text{dom}$

```
startWorker(alert);
```

$\forall f \in \tau(\text{startWorker})$
$\quad \tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{\text{dom}\}$ $\varphi(fn_2) = \{\}$

Caveat: Analysis overapproximates effects

```
alert = fn₁() fx[dom] {};

startWorker =
  fn₂(func fx[!dom]) {};

if (false) {
  startWorker(alert);
}
```

$\tau(\text{alert}) \supseteq \{fn_1\}$

$\varphi(fn_1) \supseteq \{\text{dom}\}$

$\tau(\text{startWorker}) \supseteq \{fn_2\}$

$\forall f \in \tau(\text{arg1}(fn_2))\ \varphi(f) \not\ni \text{dom}$

$\forall f \in \tau(\text{startWorker})$
$\quad \tau(\text{arg1}(f)) \supseteq \tau(\text{alert})$

$\tau(\text{alert}) = \{fn_1\}$ $\tau(\text{startWorker}) = \{fn_2\}$ $\tau(\text{arg1}(fn_2)) = \{fn_1\}$

$\varphi(fn_1) = \{\text{dom}\}$ $\varphi(fn_2) = \{\}$

Thank you!

Appendix: Type inclusion constraints

$\tau_x : x \to \tau$ (Variable) $\qquad \tau_{f,j} : \Lambda \times \mathcal{N} \to \tau$ (Argument)

$\tau_{o,p} : \Omega \times \Pi \to \tau$ (Property) $\qquad \tau_{f,R} : \Lambda \to \tau$ (Return)

$$C(\ x = \lambda_i\ \overline{y_j}.\ \{\ \overline{s}\ \text{return}\ z\ \};\ ) =$$
$$\{\tau_x \supseteq \{f\},\ \overline{\tau_{y_j} \supseteq \tau_{f,j}},\ \tau_{f,R} \supseteq \tau_z\} \cup \bigcup \overline{C(s)}$$
$$C(\ x = \{\overline{p : y}\}_o\ ;\ ) = \{\overline{\tau_{o,p} \supseteq \tau_y},\ \tau_x \supseteq \{o\}\}$$
$$C(\ x = y;\ ) = \{\tau_x \supseteq \tau_y\}$$
$$C(\ x = y(\overline{z_j});\ ) = \{\forall f \in \tau_y.\ \tau_x \supseteq \tau_{f,R},\ \overline{\tau_{f,j} \supseteq \tau_{z_j}}\}$$
$$C(\ \text{if}\ (x)\ \{\overline{s_t}\}\ \text{else}\ \{\overline{s_e}\}\ ) = \bigcup \overline{C(s_t)} \cup \bigcup \overline{C(s_e)}$$
$$C(\ x = y[z];\ ) = \{\forall o \in \tau_y.\ \forall p \in \Pi.\ \tau_x \supseteq \tau_{o,p}\}$$
$$C(\ x = y.p;\ ) = \{\forall o \in \tau_y.\ \tau_x \supseteq \tau_{o,p}\}$$
$$C(\ x[y] = z;\ ) = \{\forall o \in \tau_x\ \forall p \in \Pi.\ \tau_{o,p} \supseteq \tau_z\}$$
$$C(\ x.p = z;\ ) = \{\forall o \in \tau_x.\ \tau_{o,p} \supseteq \tau_z\}$$