Toward a Certified Information Flows Analysis for JavaScript

Martin Bodin

JSCert meeting

Motivation

Tainting, dynamic information flow

```
x = private ;
/* ... */
public = y ;
```

This has already been done for other languages, but also for JavaScript:

- *Noninterference through secure multi-execution*, Devriese and Piessens;
- *FlowFox: a Web Browser with Flexible and Precise Information Flow Control*, De Groef, Devriese, Nikiforakis and Piessens.

Direct Flows

Non-interference

If we change any private value, we don't produce any observationnal changes in the public ones.

Goal for JavaScript

Tracking direct flows (weaker than non-interference).

Undetected Flows

```
if (private)
    public = 1 ;
else public = 2 ;
```

Indirect Flows

```
if (complexFunction ())
    /* false in practise */
    public = private ;
```

Dynamic Flows

Wish List

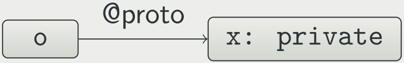
- Tracking objects.
- Prototyping.
- Functions.
- eval?
- ...

```
o = {} ;
o.x = private ;
public = o.x ;
```

Wish List

- Tracking objects.
- Prototyping.
- Functions.
- eval?
- ...

```
c = { prototype: {x: private} } ;
o = new c ;
public = o.x ;
```



Wish List

- Tracking objects.
- Prototyping.
- Functions.
- eval?
- ...

Wish List

- Tracking objects.
- Prototyping.
- Functions.
- eval?
- ...

Flow sensitive

Keep trace of time.

```
tmp = private ;
/* ... */
tmp = public ;
public = tmp ;
```

What We've Got: O'WHILE

$$
\begin{aligned}
s ::= \\
&\mid \underline{\text{skip}} \\
&\mid \underline{s_1 ; s_2} \\
&\mid \underline{\text{if } e \text{ then } s_1 \text{ else } s_2} \\
&\mid \underline{\text{while } e \text{ do } s} \\
&\mid \underline{\mathtt{x} = e} \\
&\mid \underline{e_1.\mathtt{f} = e_2} \\
&\mid \underline{\text{delete } e.\mathtt{f}}
\end{aligned}
$$

$$
\begin{aligned}
e ::= \\
&\mid \underline{c} \\
&\mid \underline{\mathtt{x}} \\
&\mid \underline{e_1 \ \mathtt{op} \ e_2} \\
&\mid \underline{\{\}} \\
&\mid \underline{e.\mathtt{f}}
\end{aligned}
$$

What's Lacking

- Closures.

In progress: closures à la λ_{JS}.

- `eval`.

Problems of lexing and parsing

Restraining it to an already parsed AST:

$$\mathtt{eval}\left(\begin{array}{ccc} & = & \\ / & & \backslash \\ \mathtt{x} & & e \end{array} \right).$$

- e1[e2].

Development of special lattices to track strings representing numbers, identifiers, etc.

What's Lacking

- Closures.

In progress: closures à la λ_{JS}.

- `eval`.

Problems of lexing and parsing

Restraining it to an already parsed AST:

$$\mathtt{eval}\left(\begin{array}{ccc} & \mathtt{=} & \\ \diagup & & \diagdown \\ \mathtt{x} & & e \end{array}\right).$$

- e1[e2].

Development of special lattices to track strings representing numbers, identifiers, etc.

What's Lacking

- Closures.

In progress: closures à la λ_{JS}.

- `eval`.

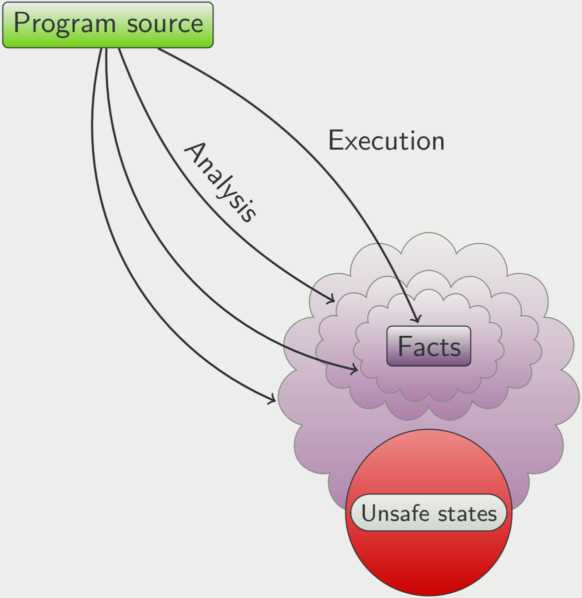
Problems of lexing and parsing

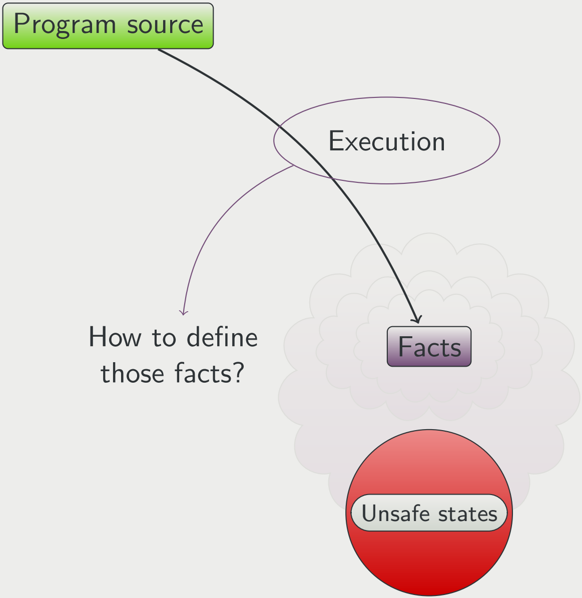
Restraining it to an already parsed AST:

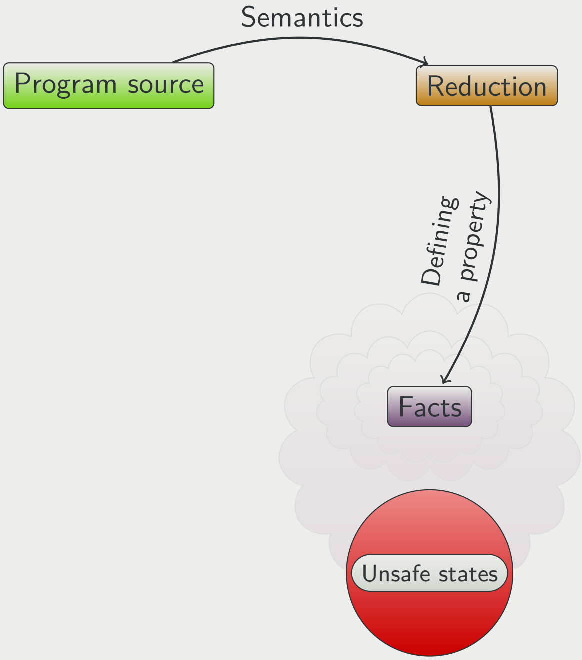
$$\texttt{eval}\left(\begin{array}{ccccc} & & \texttt{=} & & \\ & / & & \backslash & \\ \texttt{x} & & & & e \end{array}\right).$$

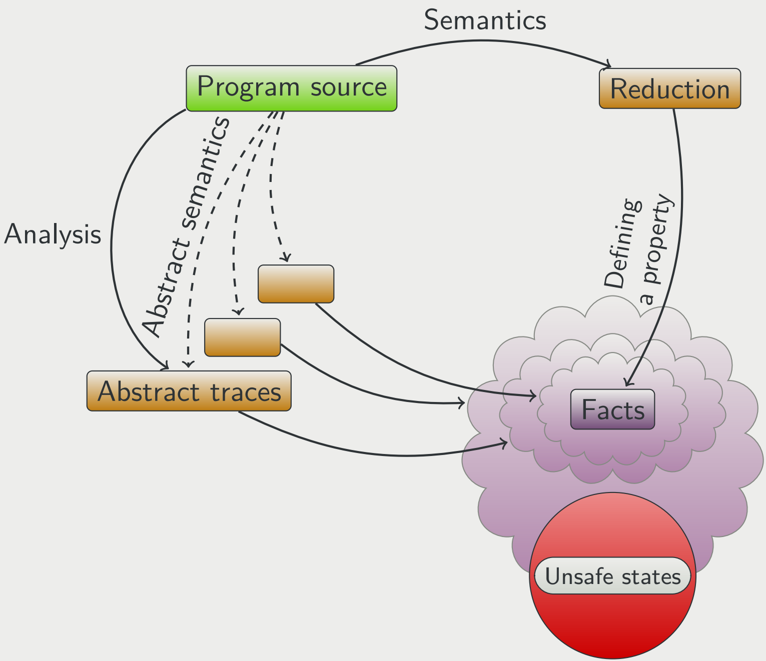
- `e1[e2]`.

Development of special lattices to track strings representing numbers, identifiers, etc.


Program source
Analysis
Execution
Facts
Unsafe states


Program source
Execution
How to define
those facts?
Facts
Unsafe states


Semantics
Program source
Reduction
Defining a property
Facts
Unsafe states


Semantics
Program source
Reduction
Analysis
Abstract semantics
Defining
a property
Abstract traces
Facts
Unsafe states

Either a S or an error.

$$\frac{S, e \to r \qquad S, \underline{\mathrm{while1}(r, e, s)} \to r'}{S, \underline{\mathrm{while}\ e\ \mathrm{do}\ s} \to r'}\ \textsc{While}$$

E, H

$$\frac{S', s \to r \qquad S', \underline{\mathrm{while2}(r, e, s)} \to r'}{S, \underline{\mathrm{while1}((S', \mathrm{true}), e, s)} \to r'}\ \textsc{WhileTrue1}$$

$$\frac{S', \underline{\mathrm{while}\ e\ \mathrm{do}\ s} \to r}{S, \underline{\mathrm{while2}(S', e, s)} \to r}\ \textsc{WhileTrue2}$$

$$\frac{}{S, \underline{\mathrm{while1}((S', \mathrm{false}), e, s)} \to S'}\ \textsc{WhileFalse}$$

$$\frac{\mathtt{abort}(t_e) = r}{S, t_e \to r}\ \textsc{Abort}$$

Instrumentation of the Semantics

Amass information from a derivation tree, making the information flow explicit (but without adding any information).

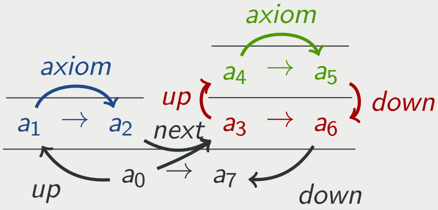
$S_\tau, t \to r$	Standard semantics
$\downarrow$	
$\tau, S_\tau, t \to \tau', r$	Adding traces
$\downarrow$	
$\tau, M_\tau, S_\tau, t \to \tau', M_{\tau'}, r$	Times of last modification
$\downarrow$	
$\tau, M_\tau, D, S_\tau, t \to \tau', M_{\tau'}, D', r$	Dependencies of expressions
$\downarrow$	
$\tau, M_\tau, D, F_\tau, S_\tau, t \to \tau', M_{\tau'}, D', F_{\tau'}, r$	Direct flows

Defining Annotations

Goal

We want this approach to scale to the full 720-rules JavaScript:

- no copy/pasting;
- a very general scheme (local rules).



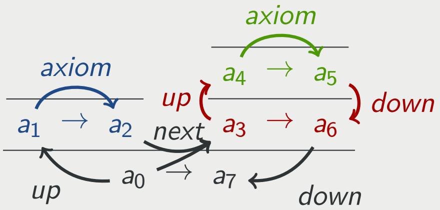
Annotations are defined directly from the semantics: we can be quite confident about them.

Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\text{ASG}\ \frac{S, e \rightarrow \quad r_0 \qquad S, \mathtt{x} =_1 r_0 \rightarrow \quad r}{S, \mathtt{x} = e \rightarrow \quad r}$$

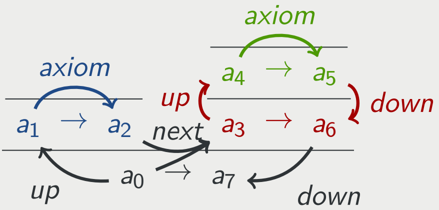


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\textsc{Asg}\ \frac{S, e \to \quad r_0 \qquad S, \mathtt{x} =_1 r_0 \to \quad r}{\tau_0, S, \mathtt{x} = e \to \quad r}$$

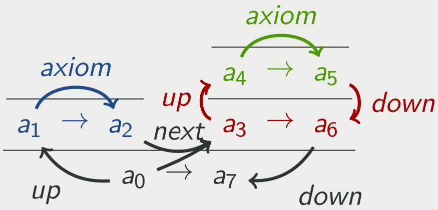


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\textsc{Asg}\ \frac{\tau_1 = \tau_0 + \textsc{AsgE} \qquad \tau_1, S, e \rightarrow r_0 \qquad S, \mathtt{x} =_1 r_0 \rightarrow r}{\tau_0, S, \mathtt{x} = e \rightarrow r}$$

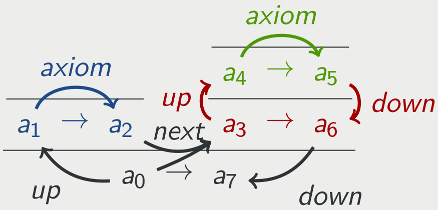


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\textsc{Asg}\ \frac{\tau_1 = \tau_0 + \textsc{AsgE} \qquad \tau_1, S, e \to \tau_2, r_0 \qquad S, \mathtt{x} =_1 r_0 \to \quad r}{\tau_0, S, \mathtt{x} = e \to \quad r}$$

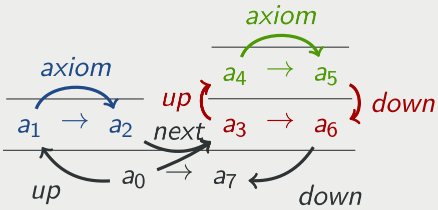


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\textsc{Asg}\ \frac{\tau_1 = \tau_0 +\!\!+\ \textsc{AsgE} \qquad \tau_3 = \tau_2 +\!\!+\ \textsc{Asg1} \qquad \tau_1, S, e \to \tau_2, r_0 \qquad \tau_3, S, \mathtt{x} =_1 r_0 \to r}{\tau_0, S, \mathtt{x} = e \to r}$$

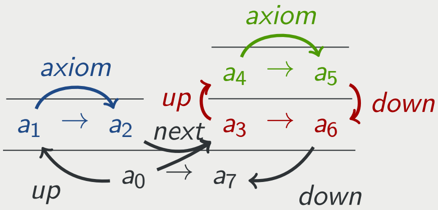


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

Flow sensitivity

$$\textsc{Asg}\ \frac{\tau_1 = \tau_0 +\!\!+\ \textsc{AsgE} \qquad \tau_3 = \tau_2 +\!\!+\ \textsc{Asg1} \qquad \tau_1, S, e \to \tau_2, r_0 \qquad \tau_3, S, \mathtt{x} =_1 r_0 \to \tau_4, r}{\tau_0, S, \mathtt{x} = e \to r}$$

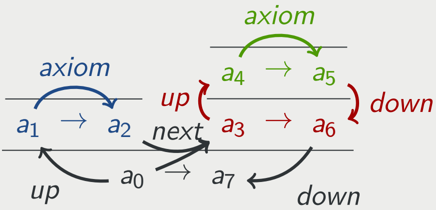


Defining Annotations: Example of Traces

Traces τ are used to identify a point in the derivation, that is to represent *execution time*.

→ Flow sensitivity

$$\textsc{Asg}\ \frac{\tau_5 = \tau_4 \mathbin{+\!\!+} \textsc{Asg} \qquad \tau_1, S, e \to \tau_2, r_0 \quad \text{with } \tau_1 = \tau_0 \mathbin{+\!\!+} \textsc{AsgE} \qquad \tau_3, S, \texttt{x} =_1 r_0 \to \tau_4, r \quad \text{with } \tau_3 = \tau_2 \mathbin{+\!\!+} \textsc{Asg1}}{\tau_0, S, \texttt{x} = e \to \tau_5, r}$$



Direct Flows

$S_\tau, t \to r$	Standard semantics
$\downarrow$	
$\tau, S_\tau, t \to \tau', r$	Adding traces τ : Traces
$\downarrow$	
$\tau, M_\tau, S_\tau, t \to \tau', M_{\tau'}, r$	Times of last modification M_τ : Var $\to$ Traces
$\downarrow$	
$\tau, M_\tau, D, S_\tau, t \to \tau', M_{\tau'}, D', r$	Dependencies of expressions $\mathtt{x}^\tau, l.\mathtt{f}^\tau, l \in D$
$\downarrow$	
$\tau, M_\tau, D, F_\tau, S_\tau, t \to \tau', M_{\tau'}, D', F_{\tau'}, r$	Direct flows $s \looparrowright t \in F_\tau$

Those flows $s \looparrowright t$ represent a dependency between:

- A **target**: a timed variable $\mathtt{x}^\tau$ or a timed field of a location $l.\mathtt{f}^\tau$.
- A **source**: a target or a location l.

Direct Flows

$S_\tau, t \to r$	Standard semantics
$\downarrow$	
$\tau, S_\tau, t \to \tau', r$	Adding traces τ : Traces
$\downarrow$	
$\tau, M_\tau, S_\tau, t \to \tau', M_{\tau'}, r$	Times of last modification M_τ : Var $\to$ Traces
$\downarrow$	
$\tau, M_\tau, D, S_\tau, t \to \tau', M_{\tau'}, D', r$	Dependencies of expressions $\mathtt{x}^\tau, l.\mathtt{f}^\tau, l \in D$
$\downarrow$	
$\tau, M_\tau, D, F_\tau, S_\tau, t \to \tau', M_{\tau'}, D', F_{\tau'}, r$	Direct flows $s \looparrowright t \in F_\tau$

Those flows $s \looparrowright t$ represent a dependency between:

- A **target**: a timed variable $\mathtt{x}^\tau$ or a timed field of a location $l.\mathtt{f}^\tau$.
- A **source**: a target or a location l.

Direct Flows

$$\textsc{Obj}\ \frac{H[l] = \bot \qquad H_1 = H[l \mapsto \{\}] \\ \tau_1 = \tau + \textsc{Obj} \qquad M_1 = M[l \mapsto \tau_1] \qquad D_1 = D \cup \{l\}}{\tau, M, D, F, E, H, \underline{\{\}} \to \tau_1, M_1, D_1, F, E, H_1, l}$$

Those flows $s \looparrowright t$ represent a dependency between:

- A **target**: a timed variable $\mathtt{x}^\tau$ or a timed field of a location $l.\mathtt{f}^\tau$.
- A **source**: a target or a location l.

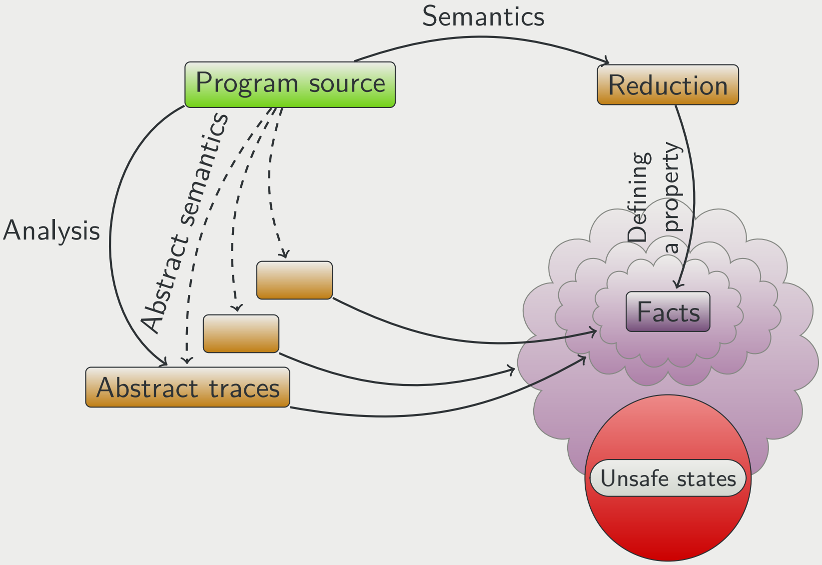
Direct Flows

$$\frac{\begin{array}{c}\tau_1 = \tau_0 + \textsc{AsgE} \qquad \tau_3 = \tau_2 + \textsc{Asg1} \\ \tau_5 = \tau_4 + \textsc{Asg} \qquad \tau_1, M, \emptyset, F_0, S, e \to \tau_2, M', D, F_1, r_0 \\ \tau_3, M', D, F_1, S, \underline{\mathtt{x} =_1 r_0} \to \tau_4, M'', \emptyset, F_2, r\end{array}}{\tau_0, M, \emptyset, F_0, S, \underline{\mathtt{x} = e} \to \tau_5, M'', \emptyset, F_2, r} \textsc{Asg}$$

$$\frac{\tau' = \tau + \textsc{Asg1} \qquad E' = E[\mathtt{x} \mapsto v] \qquad M' = M[\mathtt{x} \mapsto \tau']}{\tau, M, D, F, S, \underline{\mathtt{x} =_1 (E, H, v)} \to \tau', M', \emptyset, \left\{ d \rightsquigarrow \mathtt{x}^{\tau'} | d \in D \right\} \cup F, E', H} \textsc{Asg1}$$

Those flows $s \rightsquigarrow t$ represent a dependency between:

- A **target**: a timed variable $\mathtt{x}^\tau$ or a timed field of a location $l.\mathtt{f}^\tau$.
- A **source**: a target or a location l.


Semantics
Program source
Reduction
Analysis
Abstract semantics
Defining a property
Facts
Abstract traces
Unsafe states

Classical points-to abstractions

We chosed something very classical, checking whether it can fit.

- Objects are abstracted by their point of allocation.

$$l^\sharp \in \mathrm{Loc}^\sharp = \mathcal{P}\left(PP\right)$$

- Abstract values are abstract locations and the set of variable on which they depend.

$$v^\sharp \in \mathrm{Val}^\sharp = \mathrm{Loc}^\sharp \times \mathcal{P}\left(\mathrm{Var} \times PP\right)$$

Abstract Flows

- Environment and heap store the last program point of modification.

$$E^\sharp \in \mathrm{Env}^\sharp = \mathrm{Var} \to \left(\mathcal{P}\left(PP\right) \times \mathrm{Val}^\sharp\right)$$

$$H^\sharp \in \mathrm{Heap}^\sharp = \mathrm{Loc}^\sharp \to \mathrm{Field} \to \left(\mathcal{P}\left(PP\right) \times \mathrm{Val}^\sharp\right)$$

- This leads to the following abstract flows:

$$\mathrm{Store}^\sharp = (\mathrm{Var} \times PP) + (PP \times \mathrm{Field} \times PP) \qquad \mathrm{Source}^\sharp = PP + \mathrm{Store}^\sharp$$

$$s^\sharp \leadsto^\sharp t^\sharp \in \mathrm{Dep}^\sharp = \mathcal{P}\left(\mathrm{Source}^\sharp \times \mathrm{Store}^\sharp\right)$$

$$\frac{E^\sharp, H^\sharp, \underline{e} \rightarrow^\sharp v^\sharp, d^\sharp}{E^\sharp, H^\sharp, \underline{\mathtt{x}^p = e} \rightarrow^\sharp E^\sharp\left[\mathtt{x} \mapsto \left(\{p\},^\sharp v\right)\right], H^\sharp, \left(v^\sharp_l \cup d^\sharp\right)^\sharp \looparrowleft \mathtt{x}^p} \textsc{Asg}$$

$$\frac{\begin{array}{c} E^\sharp \sqsubseteq E^\sharp_0 \qquad H^\sharp \sqsubseteq H^\sharp_0 \\ E^\sharp_0, H^\sharp_0, \underline{s} \rightarrow^\sharp E^\sharp_1, H^\sharp_1, F^\sharp \\ E^\sharp_1 \sqsubseteq E^\sharp_0 \qquad H^\sharp_1 \sqsubseteq H^\sharp_0 \end{array}}{E^\sharp, H^\sharp, \underline{\text{while } e \text{ do } s} \rightarrow^\sharp E^\sharp_0, H^\sharp_0, F^\sharp} \textsc{While}$$

Correctness of the analysis

We define a relation $\prec$ relating

- Traces and PP;
- Heap and $\mathrm{Heap}^\sharp$;
- ...

Theorem (Work in progress)

If

$$[], \emptyset, [], \emptyset, \emptyset, \underline{s} \rightarrow \tau, M_\tau, F_\tau, E_\tau, H_\tau$$

and

$$\bot, \bot, \underline{s} \rightarrow^\sharp E^\sharp, H^\sharp, F^\sharp$$

then $E \prec E^\sharp$, $H \prec H^\sharp$ *and* $F_\tau \prec F^\sharp$.

Challenge

Fit It Into COQ.

$S_\tau, t \to r$	Standard semantics
$\downarrow$	
$\tau, S_\tau, t \to \tau', r$	Adding traces τ : Traces
$\downarrow$	
$\tau, M_\tau, S_\tau, t \to \tau', M_{\tau'}, r$	Times of last modification M_τ : Var $\to$ Traces
$\downarrow$	
$\tau, M_\tau, D, S_\tau, t \to \tau', M_{\tau'}, D', r$	Dependencies of expressions $\mathtt{x}^\tau, l.\mathtt{f}^\tau, l \in D$
$\downarrow$	
$\tau, M_\tau, D, F_\tau, S_\tau, t \to \tau', M_{\tau'}, D', F_{\tau'}, r$	Direct flows $s \rightsquigarrow t \in F_\tau$

Fit It Into Coq

```
Section LastModified.
Variable Locations : Annotations.

Definition ModifiedAnnots := annot_s_r Locations.
Record LastModifiedHeaps : Type :=
  makeLastModifiedHeaps {
    LCEnvironment : heap var ModifiedAnnots;
    LCHeap : heap loc (heap prop_name ModifiedAnnots)
  }.

Definition LastModified :=
  ConstantAnnotations LastModifiedHeaps.
```

Fit It Into Coq

```
Definition LastModifiedAxiom_s (r : LastModifiedHeaps)
  E H t o (R : red_stat Locations E H t o) :=
  let LCE := LCEnvironment r in
  let LCH := LCHeap r in
  let (_, tau) := extract_annot_s R in
  match R with
  | red_stat_ext_stat_assign_1 _ _ _ _ _ _ _ x _ _ ⇒
    let LCE' := write LCE x tau
    in makeLastModifiedHeaps LCE' LCH
  | red_stat_stat_delete _ _ _ _ _ l _ f _ _ _ _ _ _ ⇒
    let aob := read LCH l
    in let LCH' := write LCH l (write aob f tau)
    in makeLastModifiedHeaps LCE LCH'
  | red_stat_ext_stat_set_2 _ _ _ _ _ _ _ _ l _ f _ _ _ _ _ _ ⇒
    let aob := read LCH l
    in let LCH' := write LCH l (write aob f tau)
    in makeLastModifiedHeaps LCE LCH'
  | _ ⇒ makeLastModifiedHeaps LCE LCH
  end.
```

Fit It Into Coq

```
Definition annotLastModified :=
  makeIterativeAnnotations LastModified
    (init_e Transmit) (axiom_e Transmit) (up_e Transmit) (down_e
        Transmit) (next_e Transmit)
    (up_s_e Transmit) (next_e_s Transmit)
    (init_s Transmit) LastModifiedAxiom_s (up_s Transmit) (down_s
        Transmit) (next_s Transmit).

End LastModified.
```

Extensions

- Abstract domain for heap.

How to keep precision when taking as input an unknown heap?
What kind of knowledge can we assume on such a heap?

- Functions, closures: how to keep precision?
- Extensible records (how to precisely deal with prototype chains?).

$$\frac{E^\sharp, H^\sharp, \underline{e} \rightarrow^\sharp v^\sharp, d^\sharp \qquad H^\sharp[v_l^\sharp] \sqsubseteq o^\sharp \qquad o^\sharp[\mathtt{f}] = \left(p_0, v_f^\sharp\right)}{E^\sharp, H^\sharp, \underline{\text{delete } e.\mathtt{f}} \rightarrow^\sharp E^\sharp, H^\sharp, d^\sharp \hookleftarrow^\sharp v_l^\sharp.\mathtt{f}^{p_0}} \text{ DEL}$$

$$\begin{array}{rcl} \rho & ::= & \mathtt{f} : \mathit{Present}(v^\sharp) \\ & | & \mathtt{f} : \mathit{Missing}(v^\sharp) \\ & | & \mathtt{f} : \mathit{MayBeThere}(v^\sharp) \\ & | & \mathit{NoFieldsAtAll} \\ & | & \alpha \end{array}$$

Extensions

- Abstract domain for heap.

How to keep precision when taking as input an unknown heap?
What kind of knowledge can we assume on such a heap?

- Functions, closures: how to keep precision?
- Extensible records (how to precisely deal with prototype chains?).

$$\frac{E^\sharp, H^\sharp, \underline{e} \rightarrow^\sharp v^\sharp, d^\sharp \qquad H^\sharp[v_l^\sharp] \sqsubseteq o^\sharp \qquad o^\sharp[\mathtt{f}] = \left(p_0, v_f^\sharp\right)}{E^\sharp, H^\sharp, \underline{\text{delete } e.\mathtt{f}} \rightarrow^\sharp E^\sharp, H^\sharp, d^\sharp \looparrowleft^\sharp v_l^\sharp.\mathtt{f}^{p_0}} \text{ DEL}$$

$$\begin{array}{rcl} \rho & ::= & \mathtt{f} : \mathit{Present}(v^\sharp) \\ & | & \mathtt{f} : \mathit{Missing}(v^\sharp) \\ & | & \mathtt{f} : \mathit{MayBeThere}(v^\sharp) \\ & | & \mathit{NoFieldsAtAll} \\ & | & \alpha \end{array}$$

Extensions

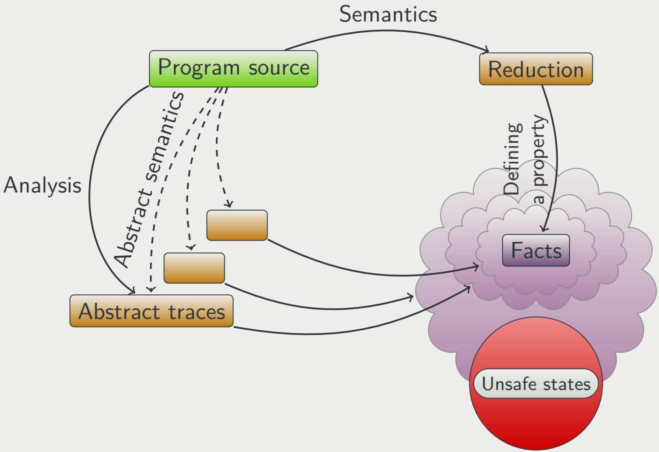
- Abstract domain for heap.

How to keep precision when taking as input an unknown heap?
What kind of knowledge can we assume on such a heap?

- Functions, closures: how to keep precision?
- Extensible records (how to precisely deal with prototype chains?).

$$\frac{E^\sharp, H^\sharp, \underline{e} \to^\sharp v^\sharp, d^\sharp \qquad H^\sharp[v_l^\sharp] \sqsubseteq o^\sharp \qquad o^\sharp[\mathtt{f}] = \left(p_0, v_f^\sharp\right)}{E^\sharp, H^\sharp, \underline{\mathrm{delete}\ \mathtt{e.f}} \to^\sharp E^\sharp, H^\sharp, d^\sharp \looparrowleft^\sharp v_l^\sharp . \mathtt{f}^{p_0}} \ \textsc{Del}$$

$$\begin{array}{rcl} \rho & ::= & \mathtt{f} : \mathit{Present}(v^\sharp) \\ & | & \mathtt{f} : \mathit{Missing}(v^\sharp) \\ & | & \mathtt{f} : \mathit{MayBeThere}(v^\sharp) \\ & | & \mathit{NoFieldsAtAll} \\ & | & \alpha \end{array}$$



Thank you for listening!