Frames and Shapes

Martin Bodin Thomas Jensen Alan Schmitt

Inria

27th of November

AJACS Meeting

JavaScript JSCert Huge semantics

Goal

A certified analyser of JavaScript.

Building Abstract Semantics from Domains, CPP'15

JSCert

Abstract Domains

Abstract Semantics

Analyser

▸ More about CPP'15

Building Abstract Semantics from Domains, CPP'15

Abstract Domains

JSCert

Abstract Semantics

Analyser

▸ More about CPP'15

Building Abstract Semantics from Domains, CPP'15

Abstract Domains

Today's Topic:
Using Separation Logic

JSCert

Abstract Semantics

Analyser

▸ More about CPP'15

Toy language

$$
\begin{array}{lll}
s ::= skip & \mid s_1;\ s_2 & \mid if\ e\ s_1\ s_2 \\
\mid while\ e\ s & \mid throw & \mid \mathtt{x} := e \\
\mid e_1.\mathtt{f} := e_2 & \mid delete\ e.\mathtt{f} &
\end{array}
$$

$$
\begin{array}{llll}
e ::= n \in \mathbb{Z} & \mid\ ? & \mid \mathtt{x} \in Var & \mid nil \\
\mid \{\} & \mid e.\mathtt{f} & \mid \mathtt{f}\ \mathtt{in}\ e & \mid \neg\ e \\
\mid = e_1\ e_2 & \mid \bowtie\ e_1\ e_2 & (\bowtie \in \{>, +, -\}) &
\end{array}
$$

- Variables and fields initially undefined, then created on the fly.
- Spurious computations throw errors.
- No prototype inheritance.
- No type conversion.
- Next move: functions.

Why Separation Logic?

Separation Logic fits JavaScript

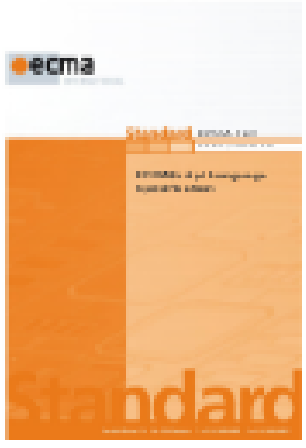
JavaScript programs use a lot of functions manipulating the heap.

- analysing each of them once for all.
- getting their footprint.

Separation Logic analyses JavaScript programs

- in a modular way.

Separation Logic fits JavaScript



From ECMAScript

A conforming implementation of ECMAScript is permitted to provide additional types, values, objects, properties, and functions beyond those described in this specification. In particular, a conforming implementation of ECMAScript is permitted to provide properties not described in this specification, and values for those properties, for objects that are described in this specification.

Separation Logic analyses JavaScript programs

- in a modular way.
- without knowing their whole environment.

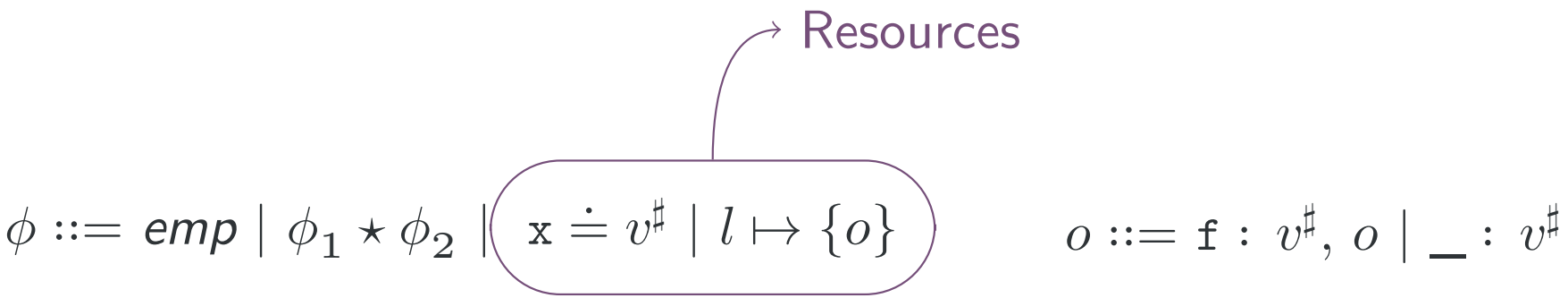
Separation Logic in One Slide

$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid \mathtt{x} \doteq v^\sharp \mid l \mapsto \{o\}$ $\qquad$ $o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$

examples

- $l_1 \mapsto \{\mathtt{f} : l_2, \_ : \top\} \star l_2 \mapsto \{\mathtt{f} : nil, \mathtt{g} : l_1 \sqcup nil, \_ : \top\}$
- $l_1 \mapsto \{\mathtt{f} : \bot, \_ : \top\} \star l_2 \mapsto \{\mathtt{g} : l_1 \sqcup nil, \_ : \top\} = False$
- $l_1 \mapsto \{\mathtt{f} : l_1, \_ : \top\} \star l_1 \mapsto \{\mathtt{g} : l_1 \sqcup nil, \_ : \top\} = False$
- $\mathtt{x} \doteq v^\sharp \star \mathtt{x} \doteq v^\sharp = False$

Separation Logic in One Slide



examples

- $l_1 \mapsto \{\mathtt{f} : l_2, \_ : \top\} \star l_2 \mapsto \{\mathtt{f} : nil, \mathtt{g} : l_1 \sqcup nil, \_ : \top\}$
- $l_1 \mapsto \{\mathtt{f} : \bot, \_ : \top\} \star l_2 \mapsto \{\mathtt{g} : l_1 \sqcup nil, \_ : \top\} = False$
- $l_1 \mapsto \{\mathtt{f} : l_1, \_ : \top\} \star l_1 \mapsto \{\mathtt{g} : l_1 \sqcup nil, \_ : \top\} = False$
- $\mathtt{x} \doteq v^\sharp \star \mathtt{x} \doteq v^\sharp = False$

The Frame Rule

$$\textsc{Frame}\ \frac{\phi, s \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, s \Downarrow^{\sharp} \phi' \star \phi_c}$$

- Everything not mentionned stay the same.
- Allows to focus on the footprint of functions.

~~The~~ One Problem of JavaScript

An Introduction to Separation Logic, by John C. Reynolds

The soundness of the frame rule is surprisingly sensitive to the semantics of our programming language. Suppose, for example, we changed the behavior of deallocation, so that, instead of causing a memory fault, `dispose x` behaved like `skip` when the value of `x` was not in the domain of the heap. Then $\{emp\}\texttt{dispose x}\{emp\}$ would be valid, and the frame rule could be used to infer $\{emp \star \texttt{x} \doteq 10\}\texttt{dispose x}\{emp \star \texttt{x} \doteq 10\}$. Then, since *emp* is a neutral element for $\star$, we would have $\{\texttt{x} \doteq 10\}\texttt{dispose x}\{\texttt{x} \doteq 10\}$, which is patently false.

- JavaScript's `delete` does exactly this.

~~The~~ One Problem of JavaScript

An Introduction to Separation Logic, by John C. Reynolds

$$\dfrac{\dfrac{\dfrac{}{\{emp\}\texttt{dispose x}\{emp\}}\ \textsc{AlreadyDisposed}}{\{emp \star \texttt{x} \doteq 10\}\texttt{dispose x}\{emp \star \texttt{x} \doteq 10\}}\ \textsc{Frame}}{\{\texttt{x} \doteq 10\}\texttt{dispose x}\{\texttt{x} \doteq 10\}}\ \textsc{Rewrite}$$

- JavaScript's `delete` does exactly this.

Solution: Dark Matter Has to be Taken Into Account

- We have to track the absence of values.
- A special abstract value $\boxtimes$.
- Some formulae:
  - *emp*
  - $l \mapsto \{\_ : \boxtimes\} \star \mathtt{x} \doteq \boxtimes$
  - $l \mapsto \{\_ : \boxtimes\} \star l \mapsto \{\_ : \boxtimes\} = False$
  - $l \mapsto \{\mathtt{f} : \boxtimes \sqcup +, \_ : \boxtimes\}$

$$\frac{}{\{\mathtt{x} \doteq v^\sharp\}\mathtt{dispose\ x}\{\mathtt{x} \doteq \boxtimes\}}\ \textsc{Disposed}$$

Getting Loop Invariant

Because programs create big structures...

```
x := nil;
while? (
   t := {};
   t.next := x;
   x := t
)
```

Smallfoot: Modular Automatic Assertion Checking with Separation Logic

Josh Berdine[1], Cristiano Calcagno[2], and Peter W. O'Hearn[3]

[1] Microsoft Research
[2] Imperial College, London
[3] Queen Mary, University of London

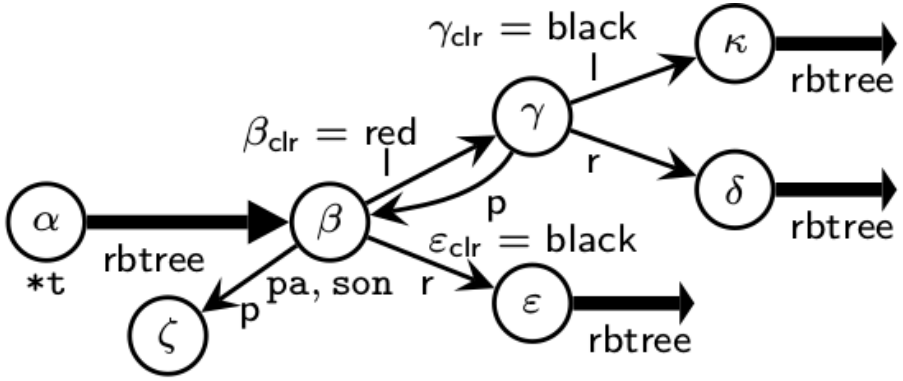
- *list*, *tree*...
- One of the first tool performing analyses using separation logic.

Relational Inductive Shape Analysis

Bor-Yuh Evan Chang
University of California, Berkeley
bec@cs.berkeley.edu

Xavier Rival
INRIA * and University of California, Berkeley
rival@di.ens.fr

- Usage of very general checkers to express structures.
- Very precise domain, but the analyser needs hints.
- No frame rule.



Towards a Program Logic for JavaScript

Philippa Gardner
Imperial College London
pg@doc.ic.ac.uk

Sergio Maffeis
Imperial College London
maffeis@doc.ic.ac.uk

Gareth Smith
Imperial College London
gds@doc.ic.ac.uk

- Understanding the prototype chains.
- $store_{LS}\,(\mathtt{x}, \mathtt{y} | \mathtt{z} : 2, \mathtt{a} : \texttt{undefined}, \mathtt{f} : L)$
- But prototype chains are not separate.
- $\phi_1 \bowtie \phi_2$.

Automatic Analysis of Open Objects in Dynamic Language Programs

Arlen Cox[1], Bor-Yuh Evan Chang[1], and Xavier Rival[2]

[1] University of Colorado Boulder, {arlen.cox,evan.chang}@colorado.edu
[2] INRIA, CNRS, ENS Paris, xavier.rival@ens.fr

- Summary nodes for JavaScript.
- No frame rule.

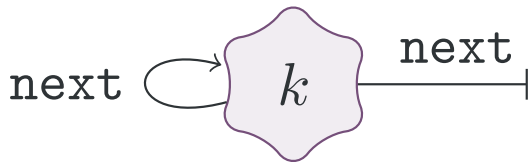
Adding Aggregates

- Looking for a simple yet generic abstraction.

```
x := nil;
while? (
    t := {};
    t.next := x;
    x := t
)
```

Adding Aggregates

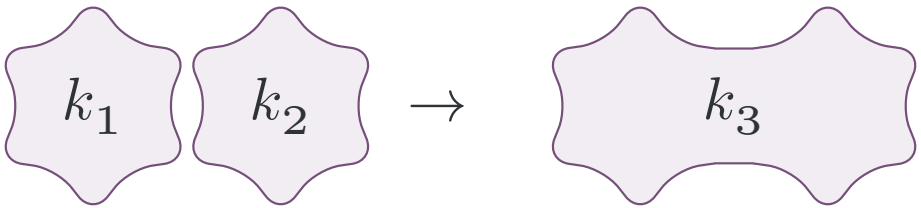
- Looking for a simple yet generic abstraction.



$$
\begin{array}{l}
\texttt{x} := nil; \\
while\,?\,(\\
\quad \texttt{t} := \{\}^{k}; \\
\quad \texttt{t.next} := \texttt{x}; \\
\quad \texttt{x} := \texttt{t} \\
)
\end{array}
$$

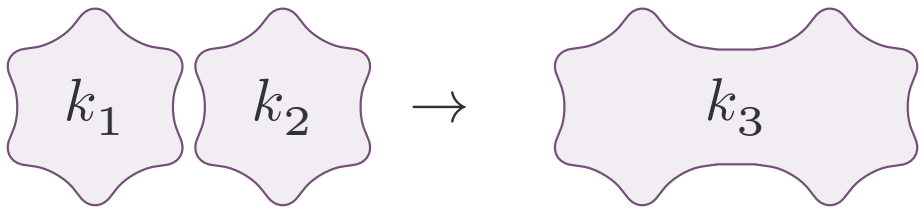
$$k \mapsto \{\texttt{next} : k \sqcup nil,\ \_ : \boxtimes\}$$

Manipulating Aggregates



$$\frac{\{\phi\}s\{k_1 \mapsto \{\texttt{f} : +, \_ : \boxtimes\} \star k_2 \mapsto \{\texttt{f} : +, \_ : \boxtimes\}\}}{\{\phi\}s\{k_3 \mapsto \{\texttt{f} : +, \_ : \boxtimes\}\}}$$

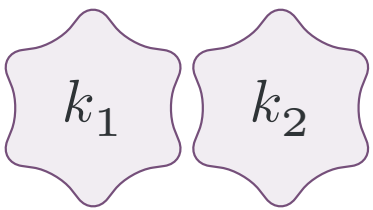
Manipulating Aggregates

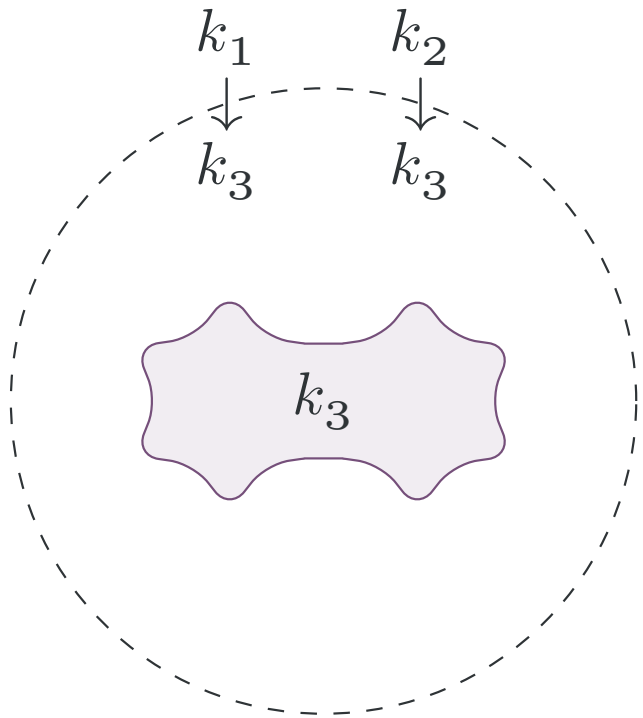


$$\frac{\{\phi\}s\{k_1 \mapsto \{\mathtt{f} : +, \_ : \boxtimes\} \star k_2 \mapsto \{\mathtt{f} : +, \_ : \boxtimes\} \star k_3 \mapsto \{\mathtt{f} : +, \_ : \boxtimes\}\}}{\{\phi\}s\{k_3 \mapsto \{\mathtt{f} : +, \_ : \boxtimes\} \star k_3 \mapsto \{\mathtt{f} : +, \_ : \boxtimes\}\}}$$

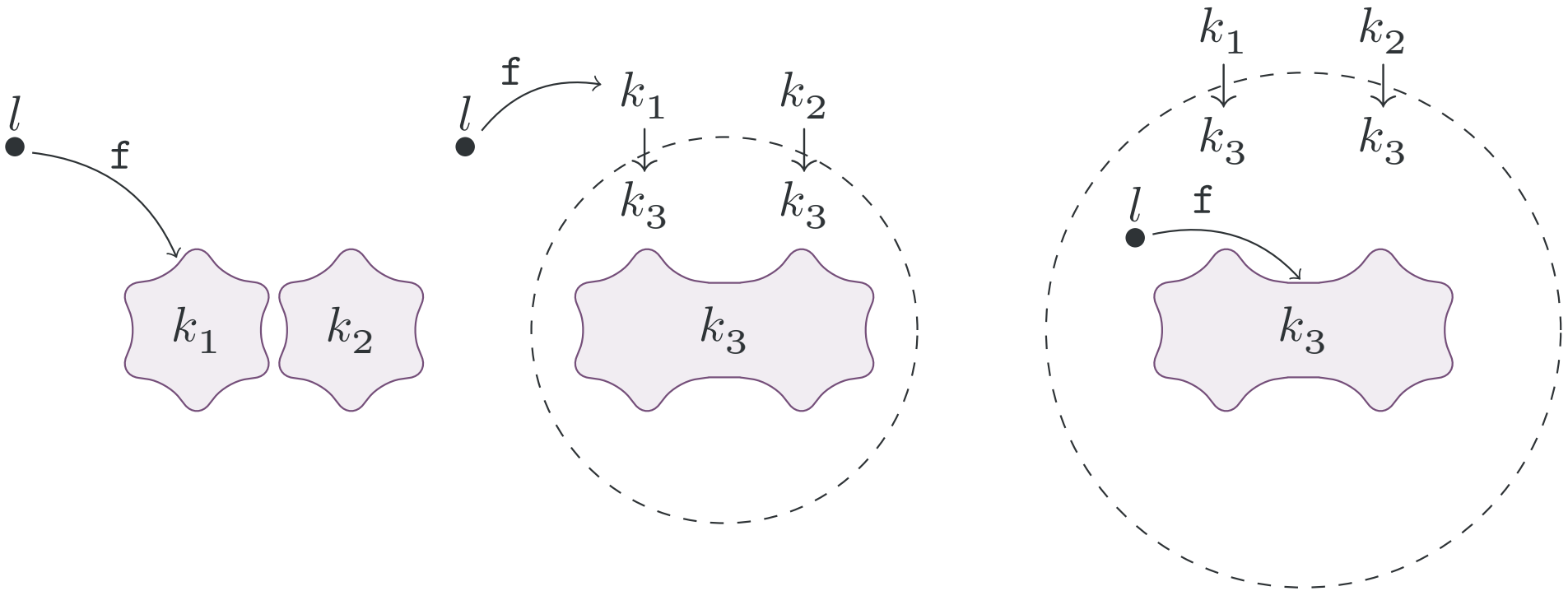
- This changes the interface!

Protecting Abstraction From the Frame Rule





Protecting Abstraction From the Frame Rule



Syntax With Membranes

$$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid \mathtt{x} \doteq v^\sharp \mid h \mapsto \{o\}$$
$$h ::= l \mid k$$

$$o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$$

$$m \in \mathfrak{M} ::= h \to h_1 + \ldots + h_n \mid \nu h$$

$$\Phi ::= (\phi \,|\, M) \qquad M \in \mathcal{P}_f(\mathfrak{M})$$

Updating the Frame Rule

The $\star$ for membraned formulae $\Phi = (\phi \,|\, M)$

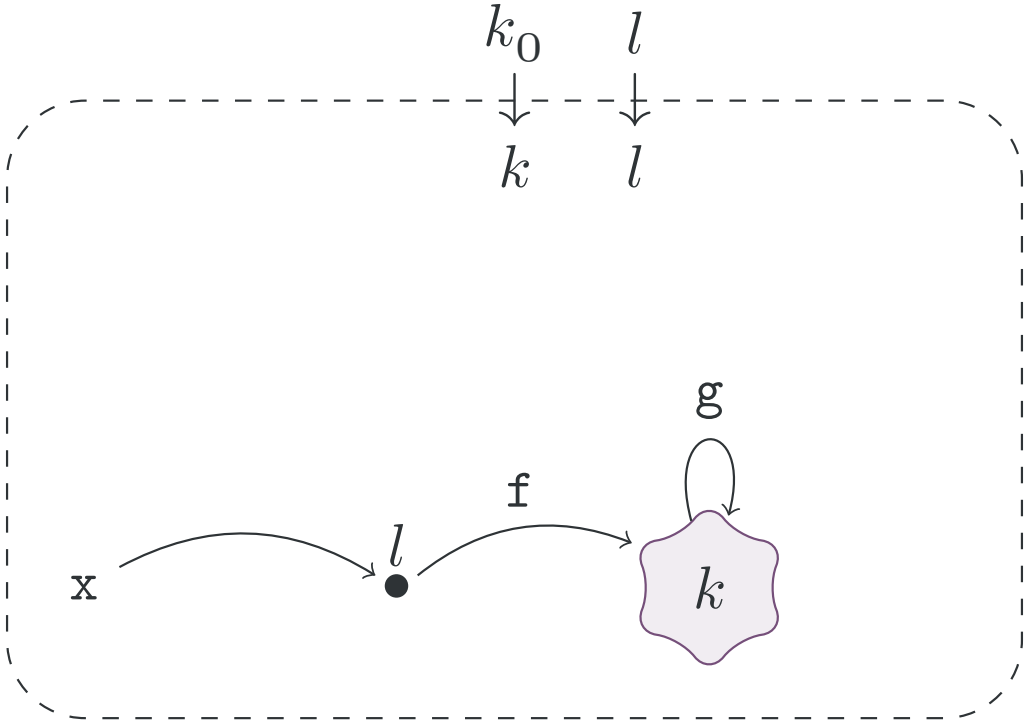
$$(\phi|\, M) \circledast (\phi_c \,|\, M_c) = \langle\!\langle\, ((\phi \,|\, M) \star \phi_c \,|\, M_c) \,\rangle\!\rangle$$

$$\textsc{Frame}\ \frac{\Phi, s \Downarrow^\sharp \Phi'}{\Phi \circledast \Phi_c, s \Downarrow^\sharp \Phi' \circledast \Phi_c}$$

▸ Correctness

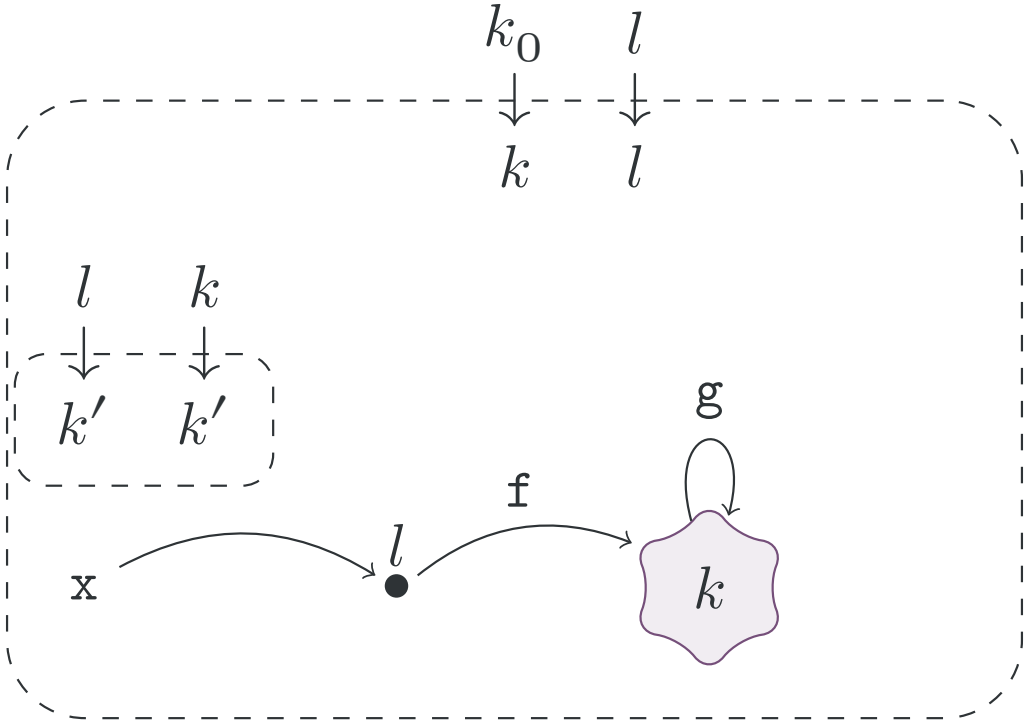
Manipulating Membranes

$$(\mathtt{x} \doteq l \star l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\} \mid k_0 \to k)$$



Manipulating Membranes

$$(\mathtt{x} \doteq l \star l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\} \,|\, k_0 \to k)$$

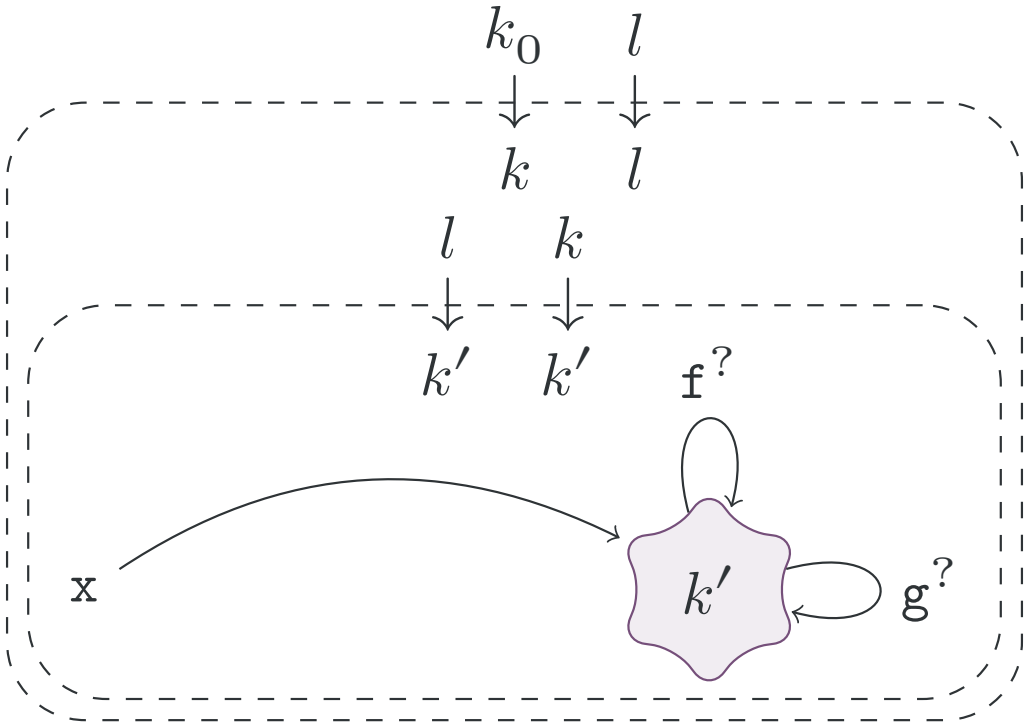


$$(\mathit{emp} \,|\, l \to k', k \to k')$$

$$\circledast\, (\mathtt{x} \doteq l \star l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\} \,|\, k_0 \to k)$$

Manipulating Membranes

$$(\mathtt{x} \doteq l \star l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\} \,|\, k_0 \to k)$$

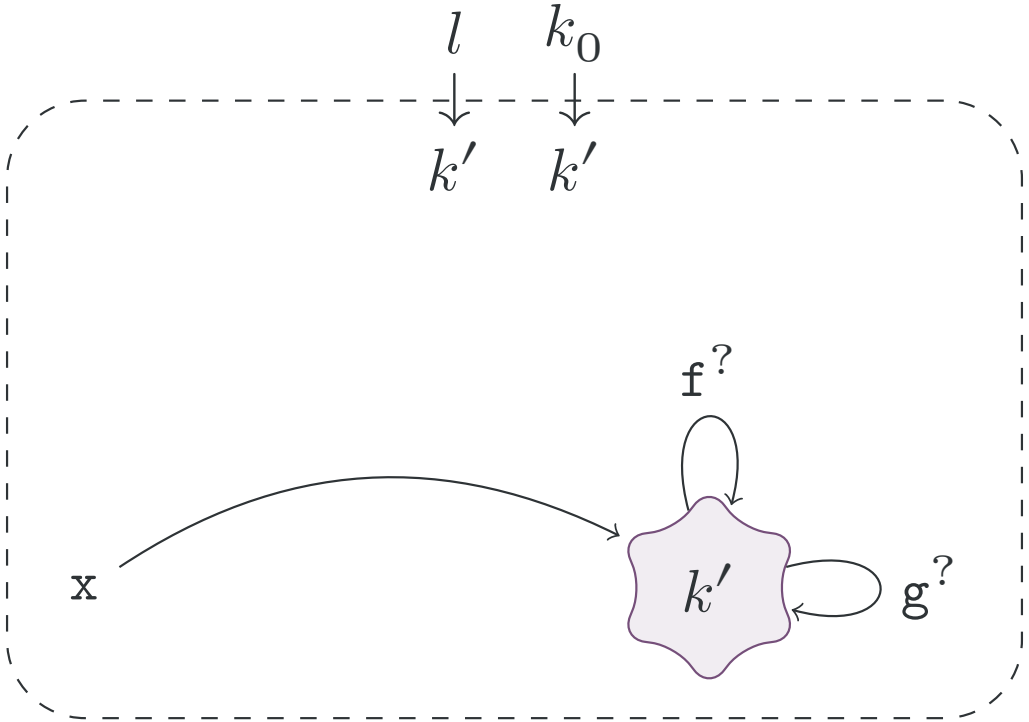


$$(\mathtt{x} \doteq k' \star k' \mapsto \{\mathtt{f} : k' \sqcup \boxtimes, \mathtt{g} : k' \sqcup \boxtimes, \_ : \boxtimes\} \,|\, k \to k', l \to k')$$

$$\circledast\, (\mathit{emp} \,|\, k_0 \to k)$$

Manipulating Membranes

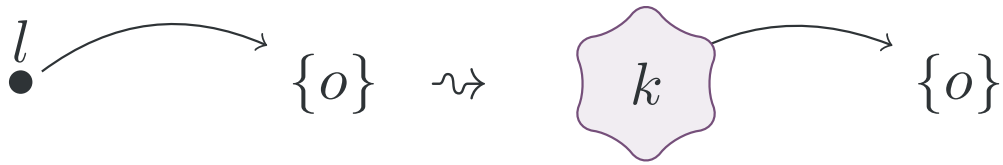
$$(\mathtt{x} \doteq l \star l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\} \mid k_0 \to k)$$



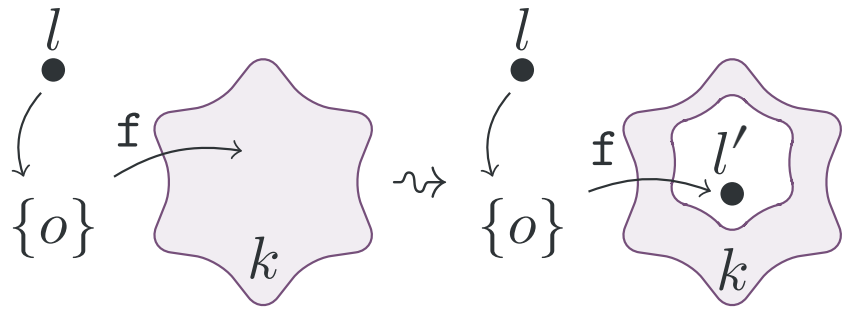
$$(\mathtt{x} \doteq k' \star k' \mapsto \{\mathtt{f} : k' \sqcup \boxtimes, \mathtt{g} : k' \sqcup \boxtimes, \_ : \boxtimes\} \mid k_0 \to k', l \to k')$$

Other Membranes Manipulations

- Transforming l into k;



- Materializing summary nodes;



- Splitting summary nodes through a filter...

Conditions

- Being correct (with respect to formulae's concretion).
- Keeping the interface safe (with respect to the frame rule).

Conclusion and Future Works

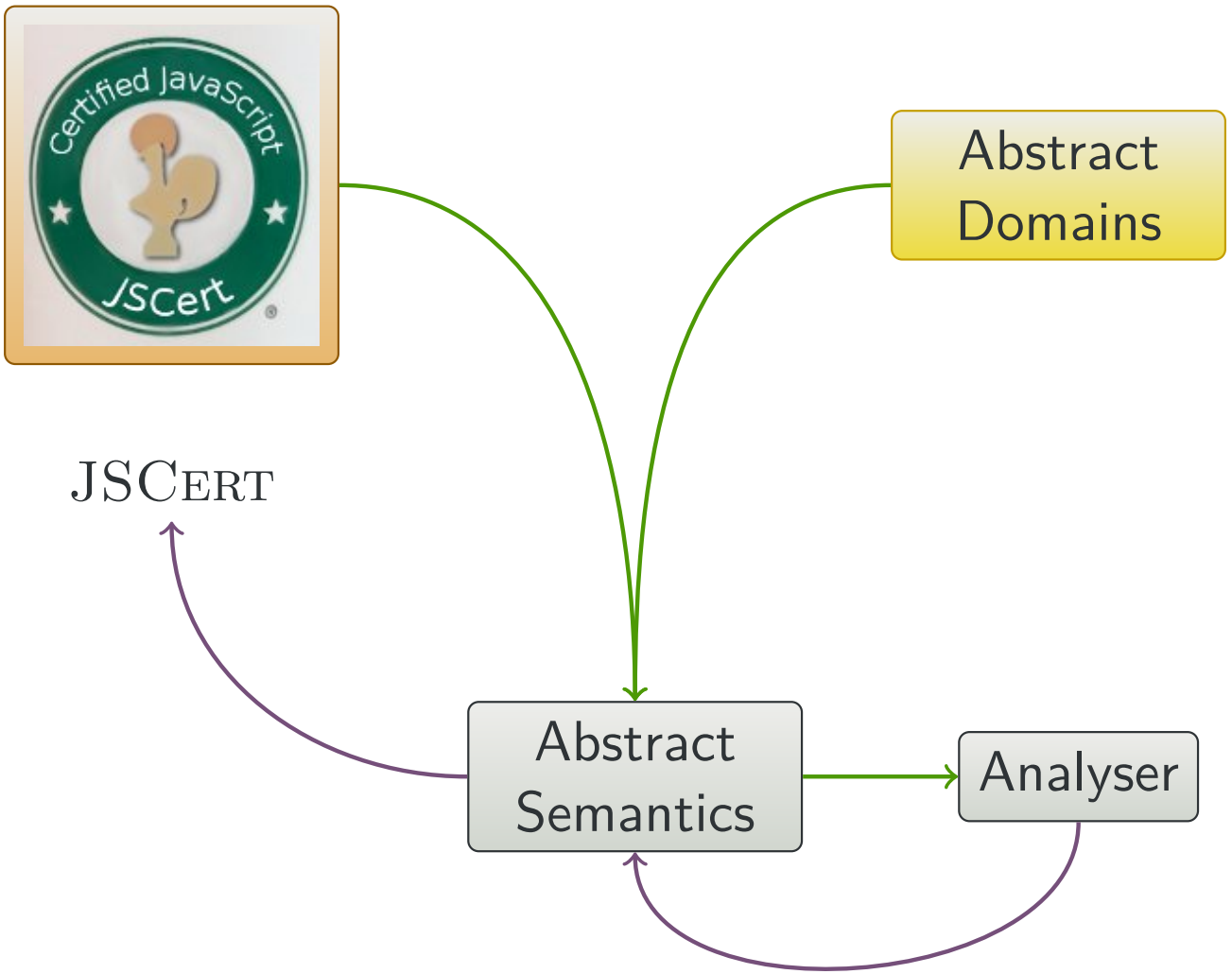
- We have a domain for JavaScript.
- Compatible with separation logic, including the frame rule.
- Compatible with the abstract interpretation framework[1]

What's next?

- Do the Coq.
- Create an analyser for a small language and see if it works.
- Scale up to JavaScript.

[1] This was not part of today's talk.

Thanks You for Listening!



Any questions?

1. Introduction

2. Separation Logic

3. Adding Shapes

CPP'15 in a Nutshell: Abstract Rules

$$\textsc{IfTrue}\ \frac{E, s_1 \Downarrow E'}{(v, E), if\ s_1\, s_2 \Downarrow E'}\ v \in \mathbb{Z}^\star \qquad \textsc{IfFalse}\ \frac{E, s_2 \Downarrow E'}{(v, E), if\ s_1\, s_2 \Downarrow E'}\ v \in \{0\}$$

$$\textsc{IfTrue}\ \frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{(v^\sharp, E^\sharp), if\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}\ \gamma(v^\sharp) \cap \mathbb{Z}^\star \neq \emptyset \qquad \textsc{IfFalse}\ \frac{E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{(v^\sharp, E^\sharp), if\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}\ \gamma(v^\sharp) \cap \{0\} \neq \emptyset$$

Back

CPP'15 in a Nutshell: Abstract Semantics

But we don't define $\Downarrow$ and $\Downarrow^\sharp$ the same way from the rules!

Concrete Semantics $\Downarrow$	Abstract Semantics $\Downarrow^\sharp$
At each step, apply <u>one</u> rule that applies	At each step, apply <u>all</u> the rules that apply

$$\frac{E_0^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \quad \uparrow \textsc{IfTrue} \qquad E_0^\sharp, s_2 \Downarrow^\sharp E_2^\sharp \quad \uparrow \textsc{IfFalse}}{\left(v^\sharp, E_0^\sharp\right), if\ s_1\, s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}$$

Back

CPP'15 in a Nutshell: Abstract Semantics

But we don't define $\Downarrow$ and $\Downarrow^\sharp$ the same way from the rules!

Concrete Semantics $\Downarrow$	Abstract Semantics $\Downarrow^\sharp$
At each step, apply <u>one</u> rule that applies	At each step, apply <u>all</u> the rules that apply
Inductive interpretation of the rules $\Downarrow = lfp\,(\mathcal{F})$	Co-inductive interpretation of the rules $\Downarrow^\sharp = gfp\,(\mathcal{F}^\sharp)$

$$\frac{E_0^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \quad \uparrow \textsc{IfTrue} \qquad E_0^\sharp, s_2 \Downarrow^\sharp E_2^\sharp \quad \uparrow \textsc{IfFalse}}{\left(v^\sharp, E_0^\sharp\right), if\ s_1\, s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}$$

Back

CPP'15 in a Nutshell: Abstract Semantics

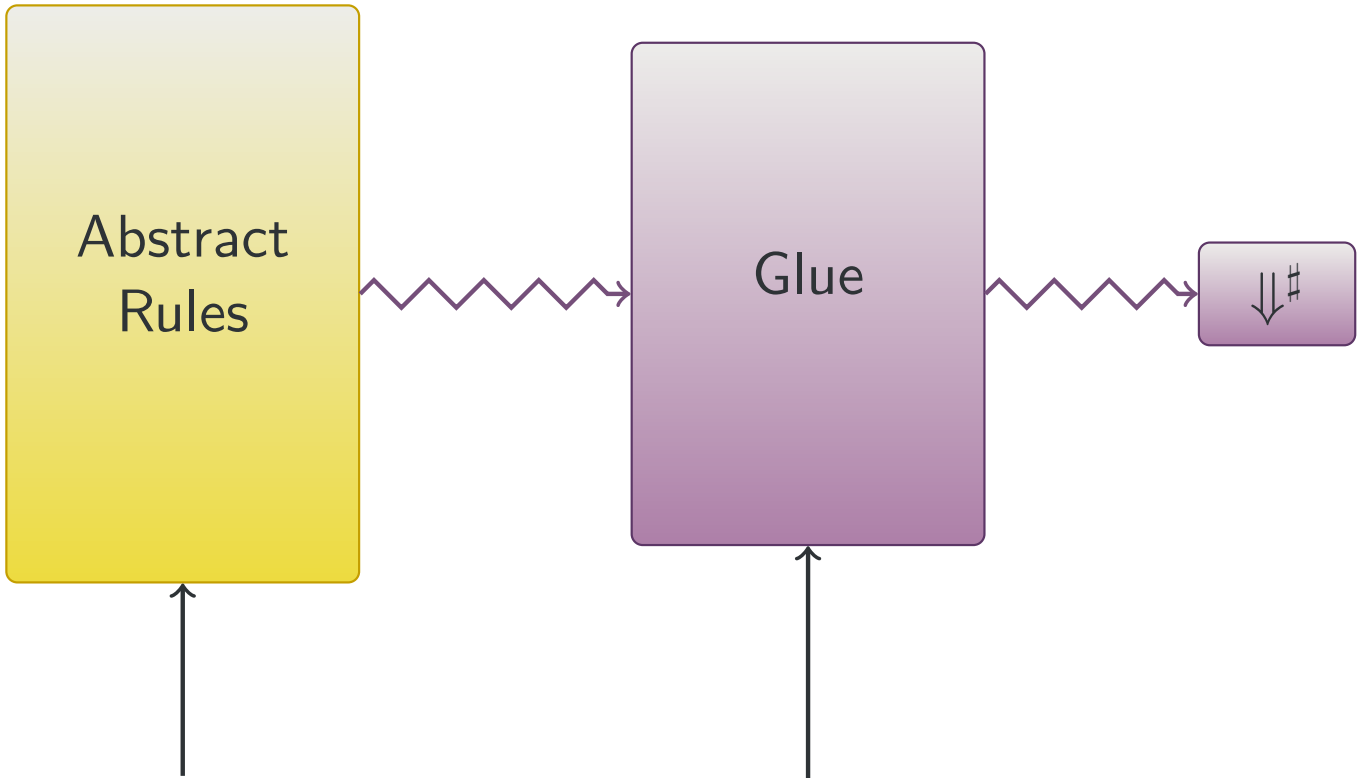
But we don't define $\Downarrow$ and $\Downarrow^\sharp$ the same way from the rules!

Concrete Semantics $\Downarrow$	Abstract Semantics $\Downarrow^\sharp$
At each step, apply <u>one</u> rule that applies	At each step, apply <u>all</u> the rules that apply
Inductive interpretation of the rules $\Downarrow = lfp\,(\mathcal{F})$	Co-inductive interpretation of the rules $\Downarrow^\sharp = gfp\,(\mathcal{F}^\sharp)$
	Allow approximations

$$\textsc{Weaken}\quad \frac{P' \sqsubseteq P \qquad \{P\}\,p\,\{Q\} \qquad Q \sqsubseteq Q'}{\{P'\}\,p\,\{Q'\}}$$

◂ Back

CPP'15 in a Nutshell



$$\textsc{IfTrue}\ \frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{\left(v^\sharp, E^\sharp\right), if\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}$$

$$\gamma\left(v^\sharp\right) \cap \mathbb{Z}^\star \neq \emptyset$$

$$\textsc{Weaken}\ \frac{P' \sqsubseteq P \qquad \{P\}\, p\, \{Q\} \qquad Q \sqsubseteq Q'}{\{P'\}\, p\, \{Q'\}}$$

Back

CPP'15 in a Nutshell: the Theorem

Theorem (Correctness)

Let t a term, σ and $\sigma^\sharp$ a concrete and an abstract semantic contexts, and r and $r^\sharp$ a concrete and an abstract results.

$$\textit{If} \begin{cases} \sigma \in \gamma\left(\sigma^\sharp\right) \\ \sigma, t \Downarrow r \\ \sigma^\sharp, t \Downarrow^\sharp r^\sharp \end{cases} \quad \textit{then } r \in \gamma\left(r^\sharp\right).$$

◂ Back

Demo?

◂ Back

1. Introduction

2. Separation Logic

3. Adding Shapes

Correctness

$$\textsc{Frame}\ \frac{\Phi, s \Downarrow^\sharp \Phi'}{\Phi \circledast \Phi_c, s \Downarrow^\sharp \Phi' \circledast \Phi_c} \qquad \textsc{Weaken}\ \frac{\Phi_1' \preccurlyeq \Phi_1 \qquad \Phi_1, s \Downarrow^\sharp \Phi_2 \qquad \Phi_2' \preccurlyeq \Phi_2}{\Phi_1', s \Downarrow^\sharp \Phi_2'}$$

Correctness = Local Correctness (in frame) +

$$\forall \Phi_1, \Phi_2.\ \Phi_1 \preccurlyeq \Phi_2 \implies \forall \Phi_c.\ \gamma\left(\Phi_1 \circledast \Phi_c\right) \subseteq \gamma\left(\Phi_2 \circledast \Phi_c\right)$$

◂ Back

1. Introduction

2. Separation Logic

3. Adding Shapes