Certified Semantics and Analysis of JavaScript

Martin Bodin

25th of November

Supervised by Alan Schmitt and Thomas Jensen

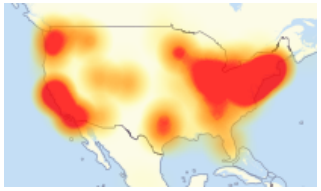
Bugs are Everywhere

- *Airbus issues software bug alert after fatal plane crash* — The Guardian, 20th of May 2015

- *Knight Capital Says Trading Glitch Cost It $440 Million* — The New York Times, 2nd of August 2012

- *Computing glitch may have doomed Mars lander* — Nature, 25th of October 2016

- *2016 Dyn cyberattack* — Wikipedia



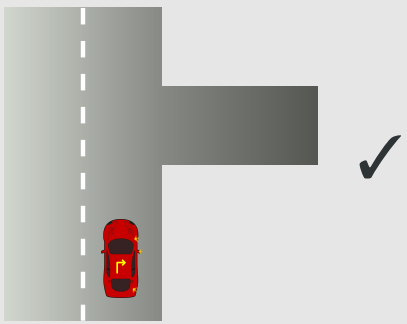
"a botnet coordinated through a large number of *Internet of Things* devices, including cameras, residential gateways, and baby monitors"

What Is a Bug?

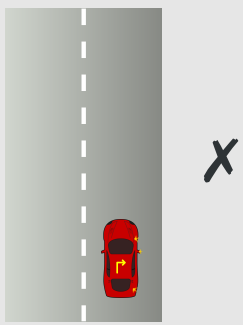
- A bug is an unwanted behaviour.

```
1 if (can_turn_right ())
2   turn_right ()
3 else
4   turn_right ()
```

Expected behaviour



Unwanted behaviour



Bugs Yield Unwanted States

Program

```
1  if (can_turn_right ())
2    turn_right ()
3  else
4    turn_right ()
```

Specification

The car does not go out of the road.

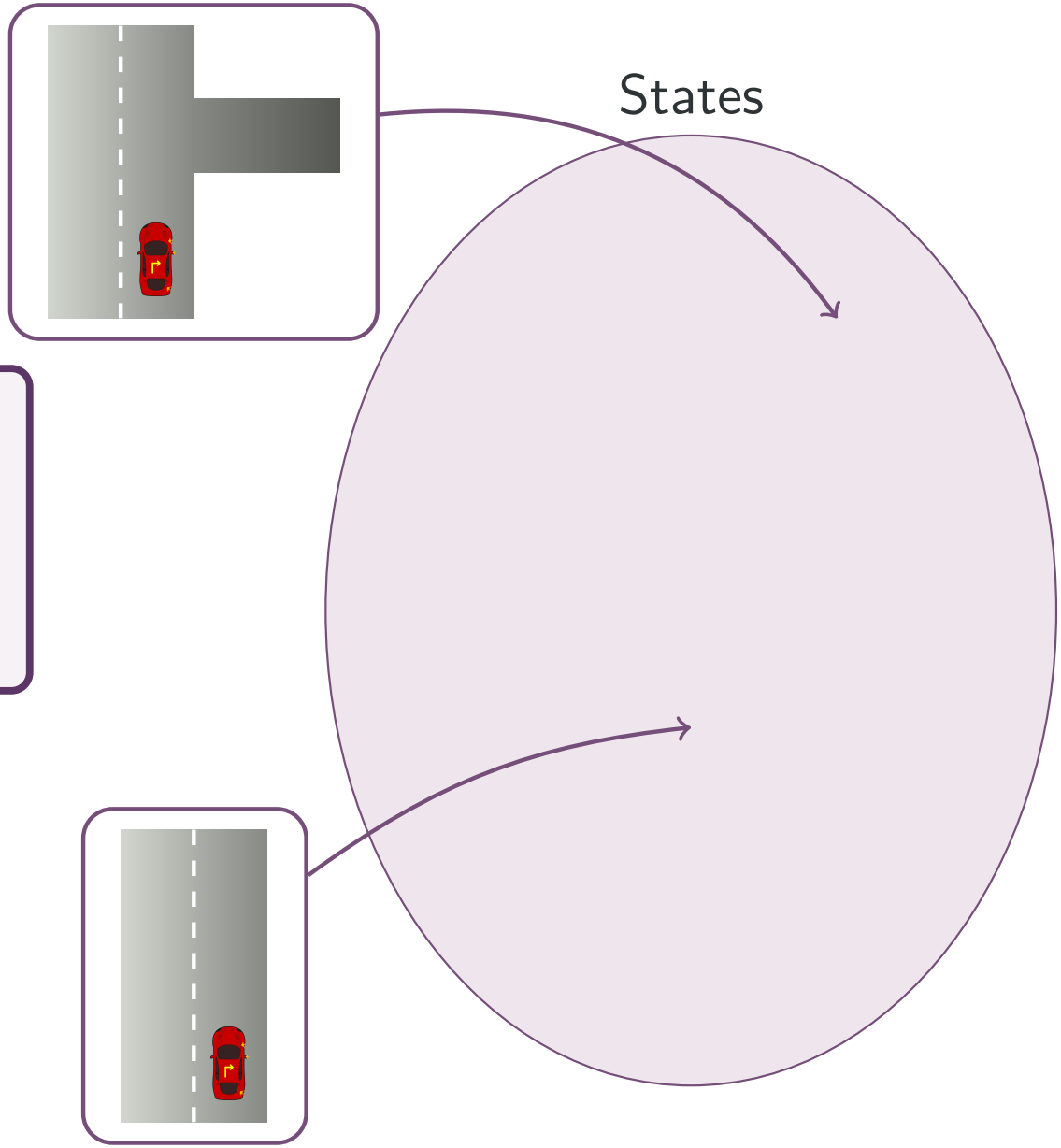
Bugs Yield Unwanted States

Program

```
1  if (can_turn_right ())
2    turn_right ()
3  else
4    turn_right ()
```

Specification

The car does not go out of the road.



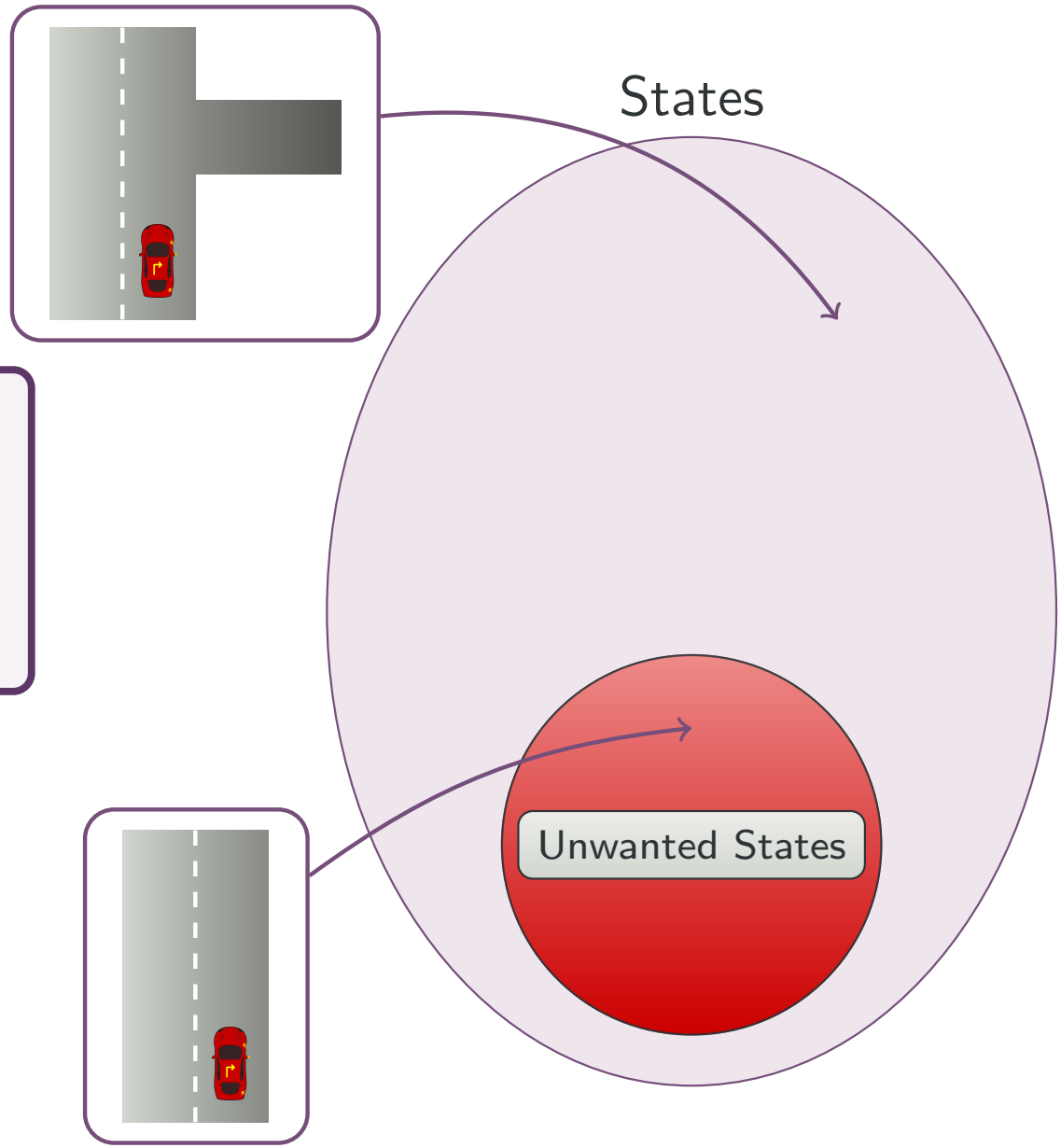
Bugs Yield Unwanted States

Program

```
1 if (can_turn_right ())
2   turn_right ()
3 else
4   turn_right ()
```

Specification

The car does not go out of the road.



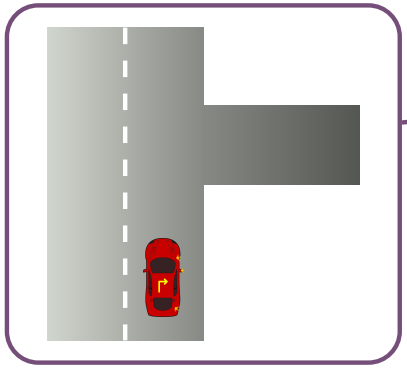
Bugs Yield Unwanted States

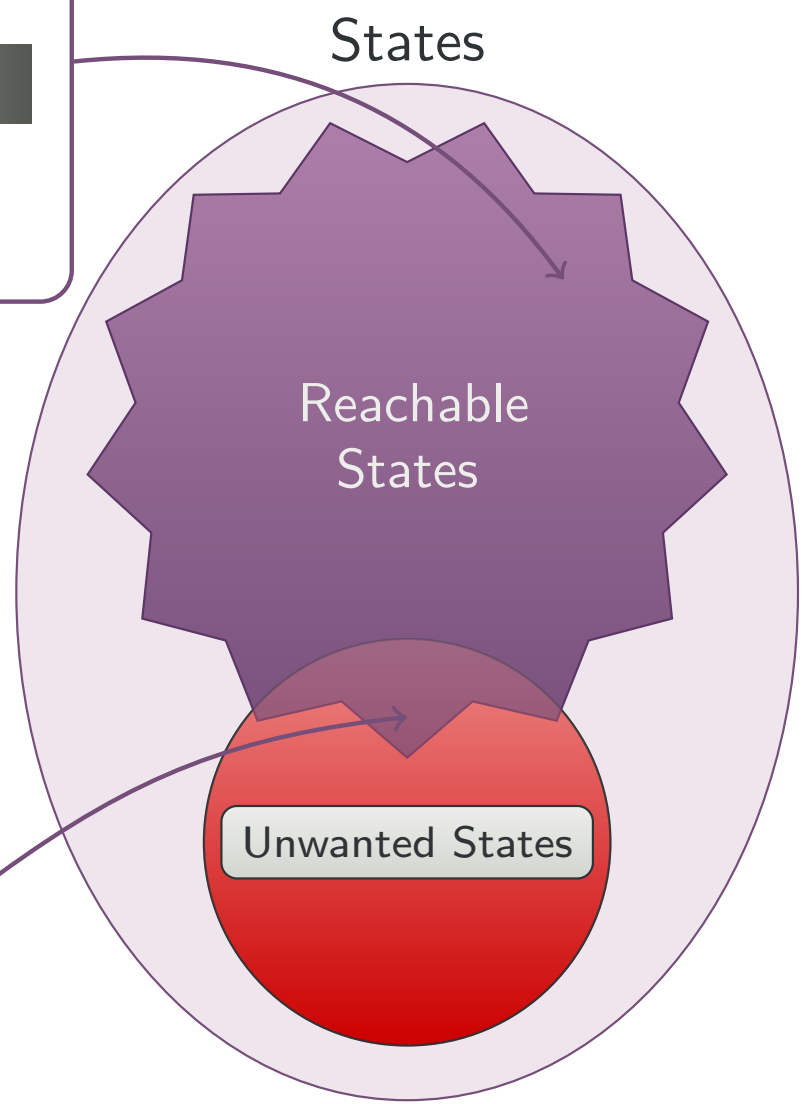
Program

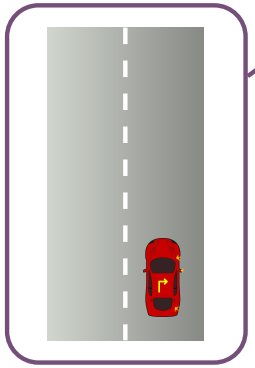
```
1  if (can_turn_right ())
2    turn_right ()
3  else
4    turn_right ()
```

Specification

The car does not go out of the road.







Bugs Yield Unwanted States

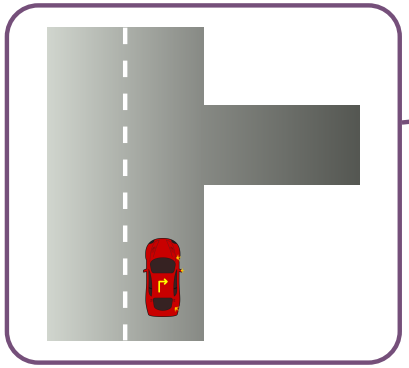
Program

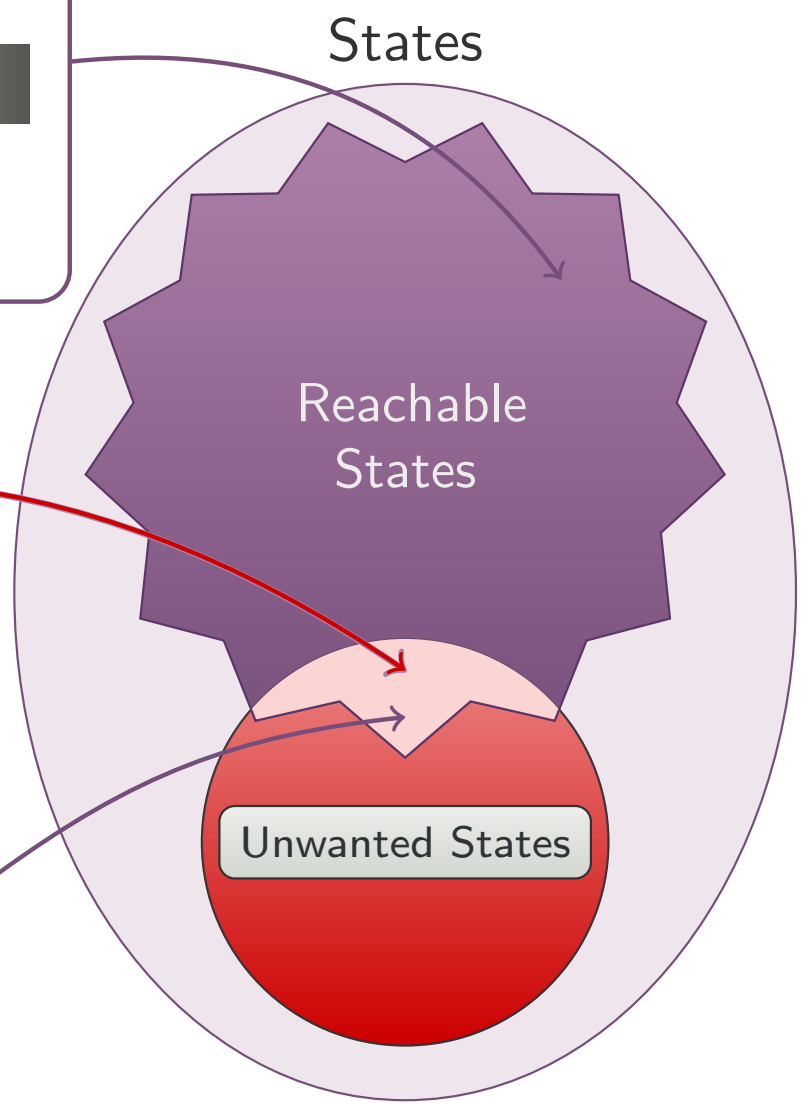
```
1  if (can_turn_right ())
2    turn_right ()
3  else
4    turn_right ()
```

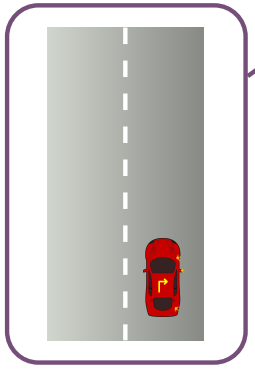
Bug

Specification

The car does not go out of the road.





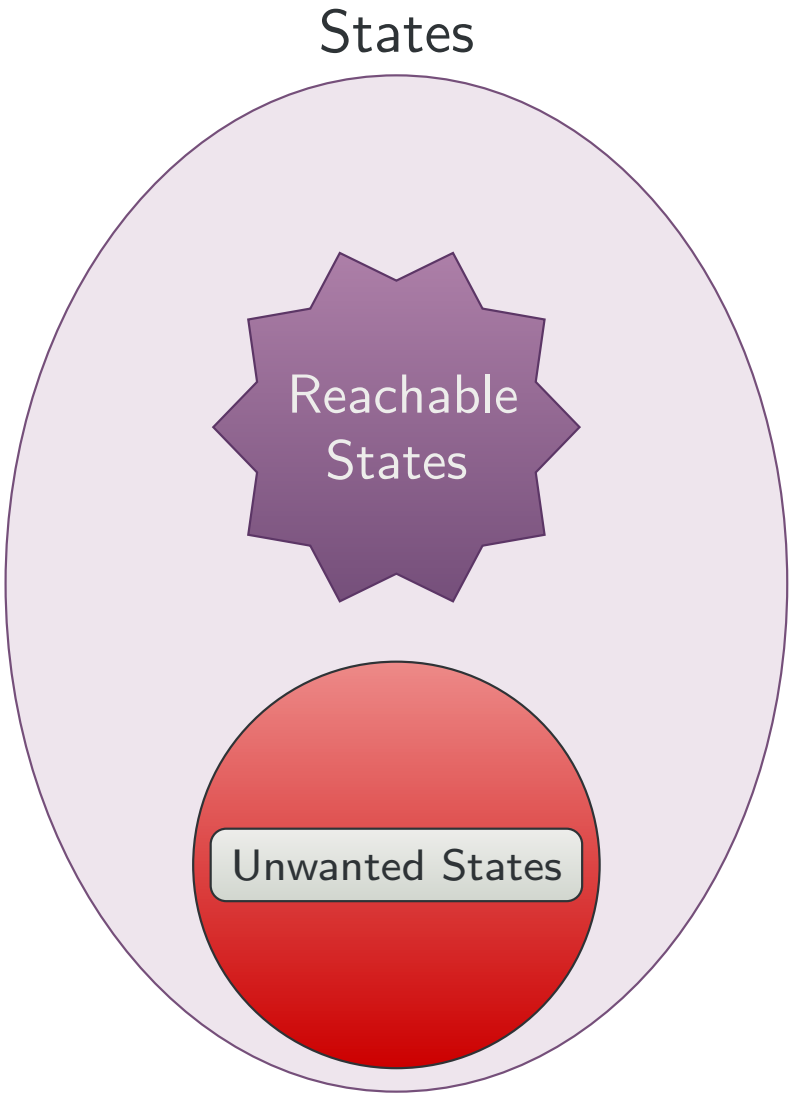


Are specifications correctly implemented?

How can we avoid bugs?

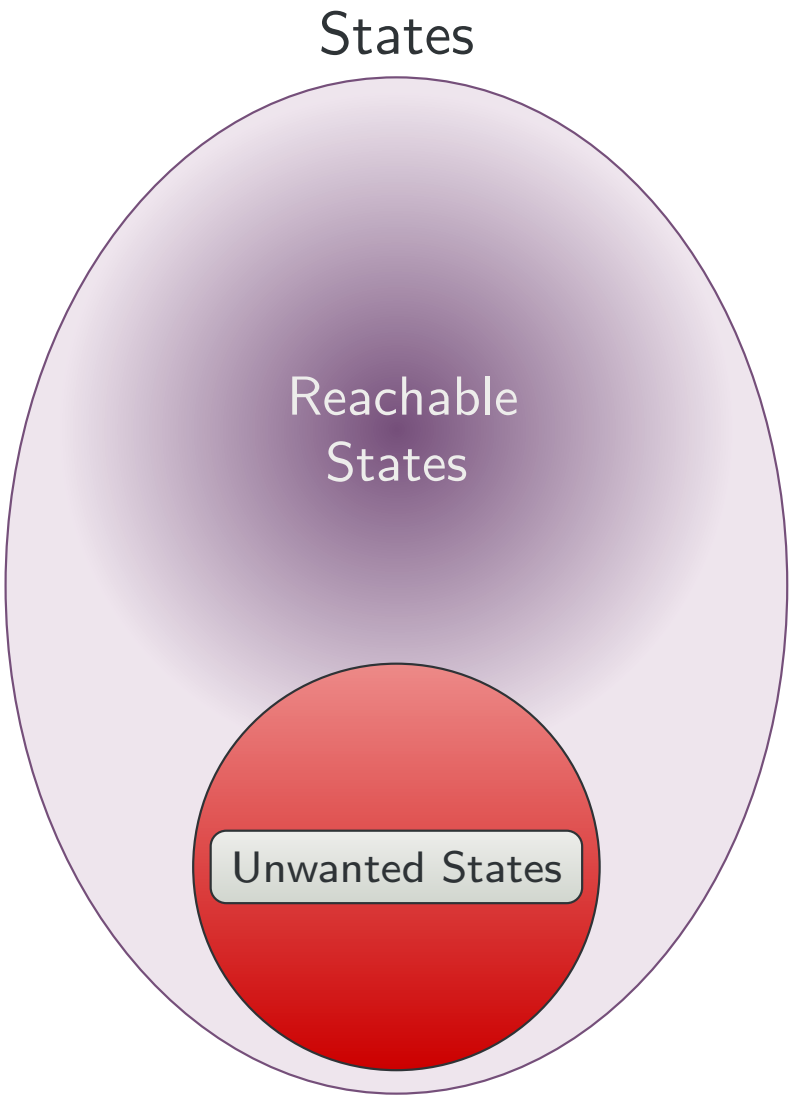
Trusting Programs: Existing Methods

Program



Trusting Programs: Existing Methods

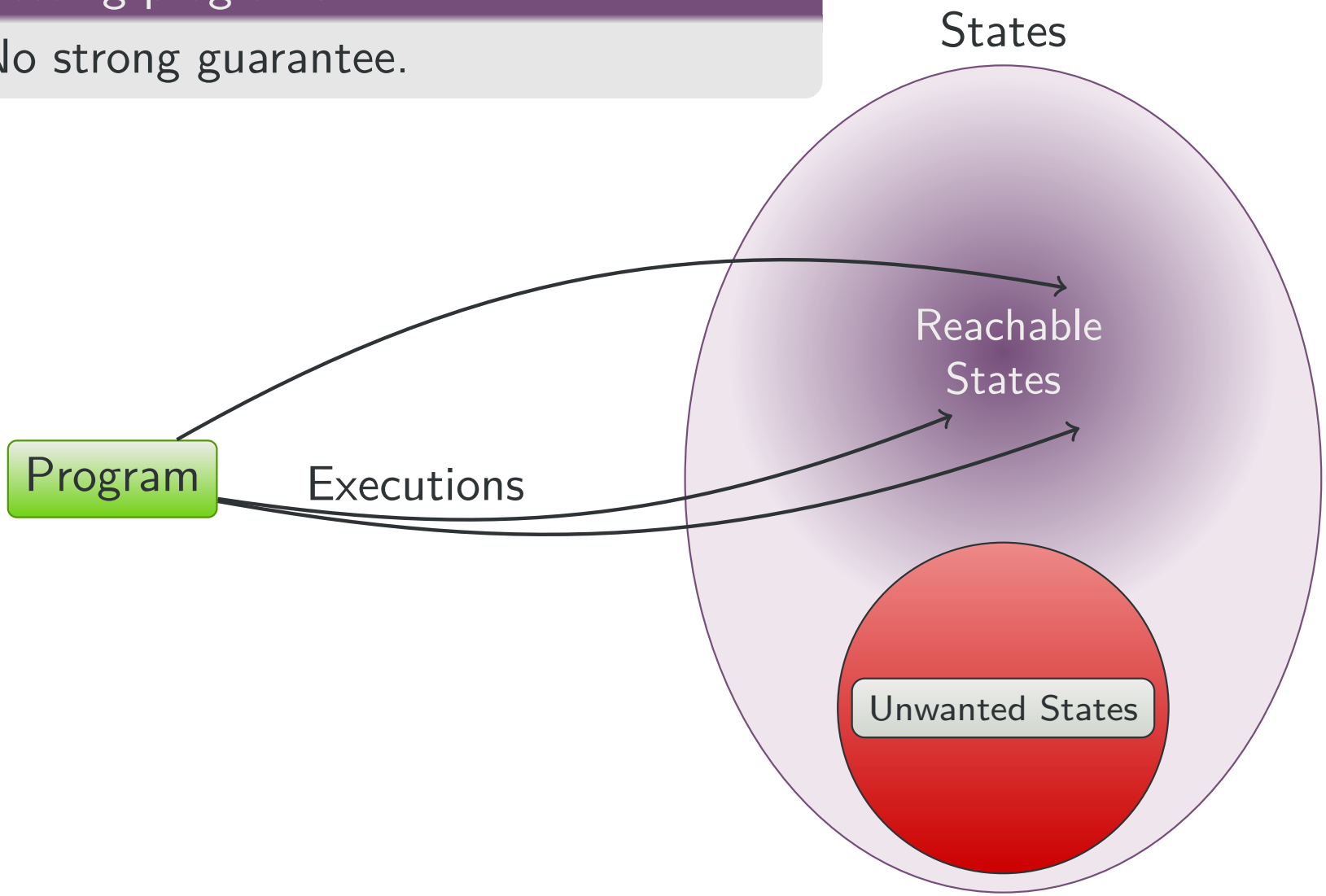
Program



Trusting Programs: Existing Methods

Testing programs

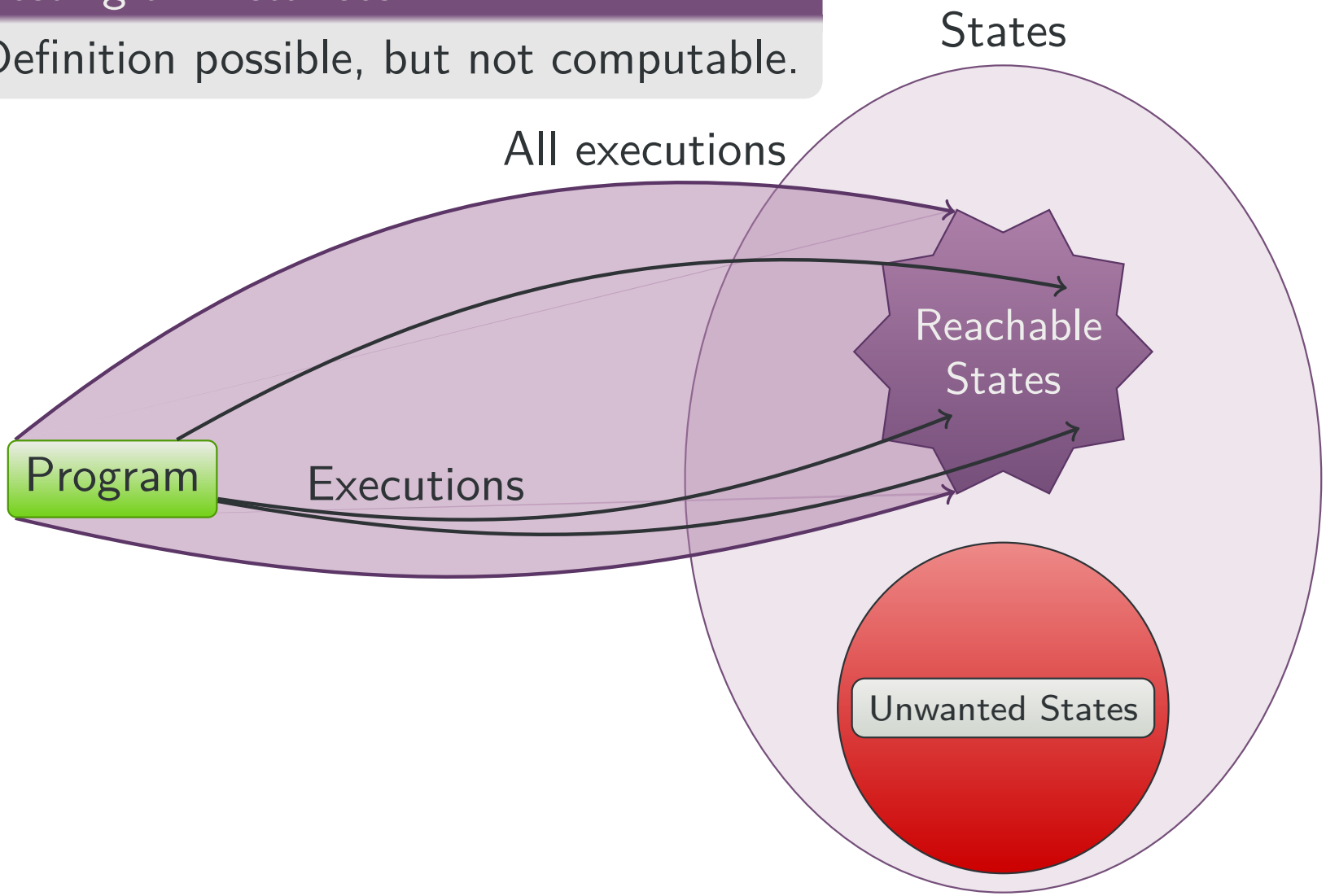
No strong guarantee.



Trusting Programs: Existing Methods

Testing *all* instances

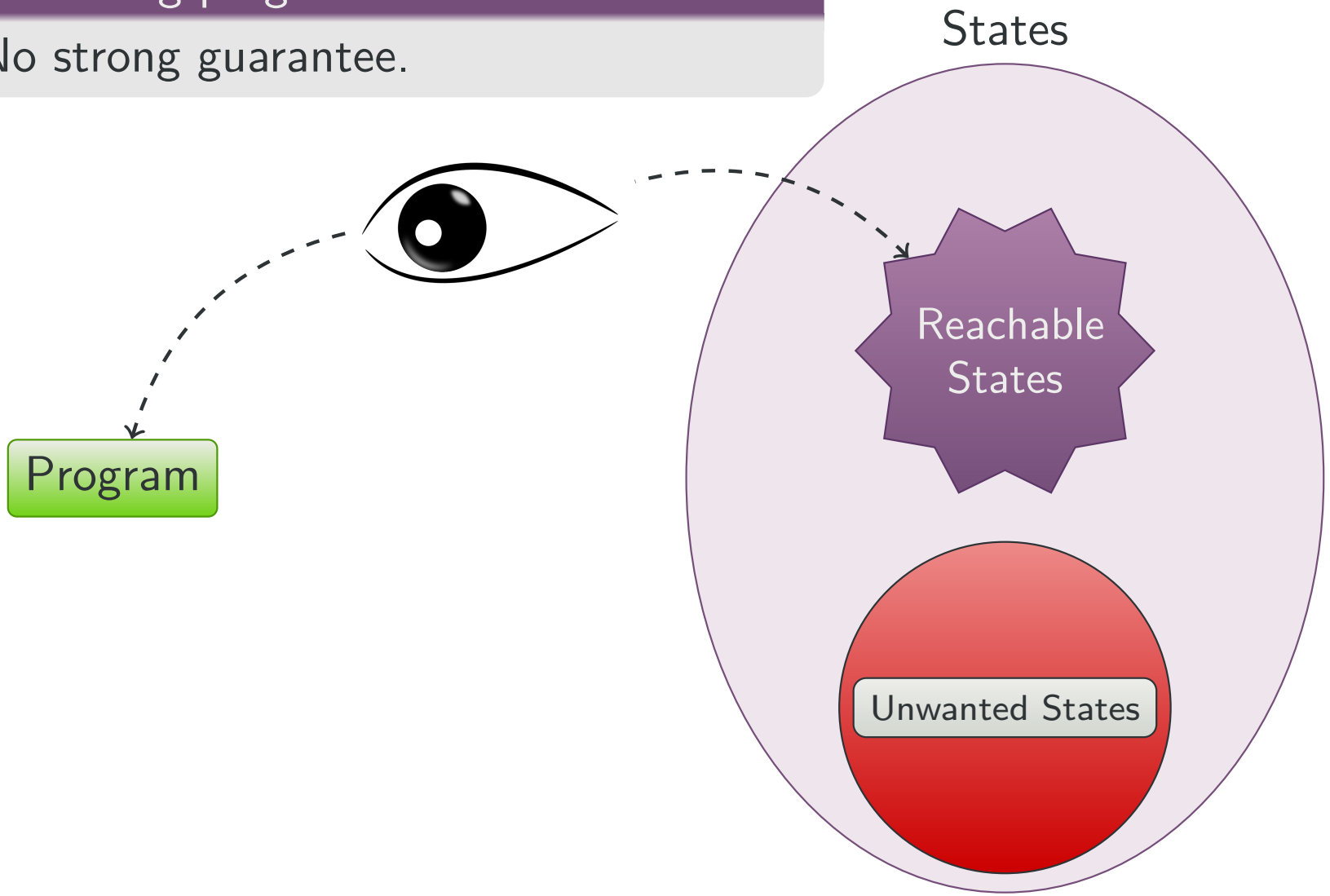
Definition possible, but not computable.



Trusting Programs: Existing Methods

Reviewing programs

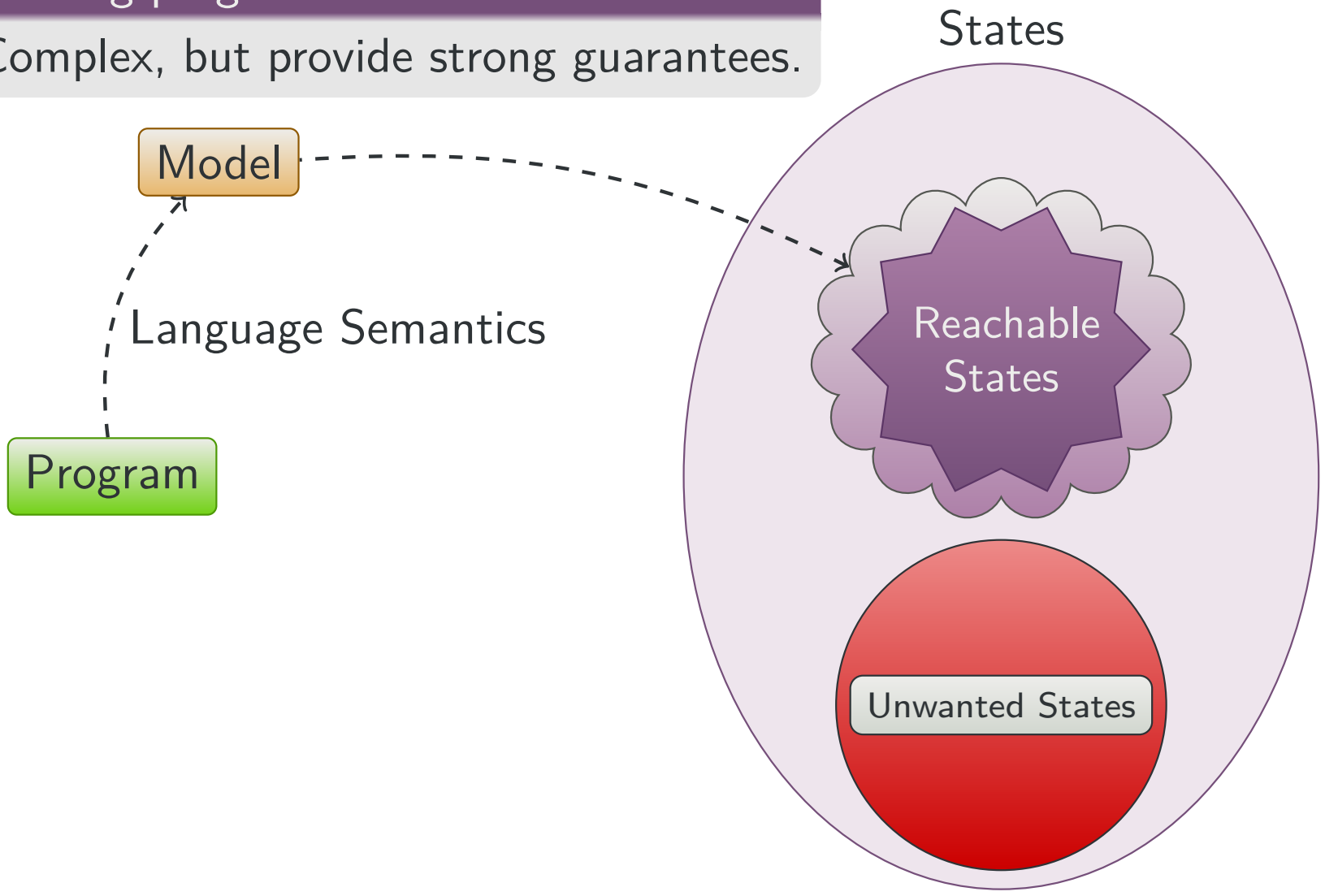
No strong guarantee.



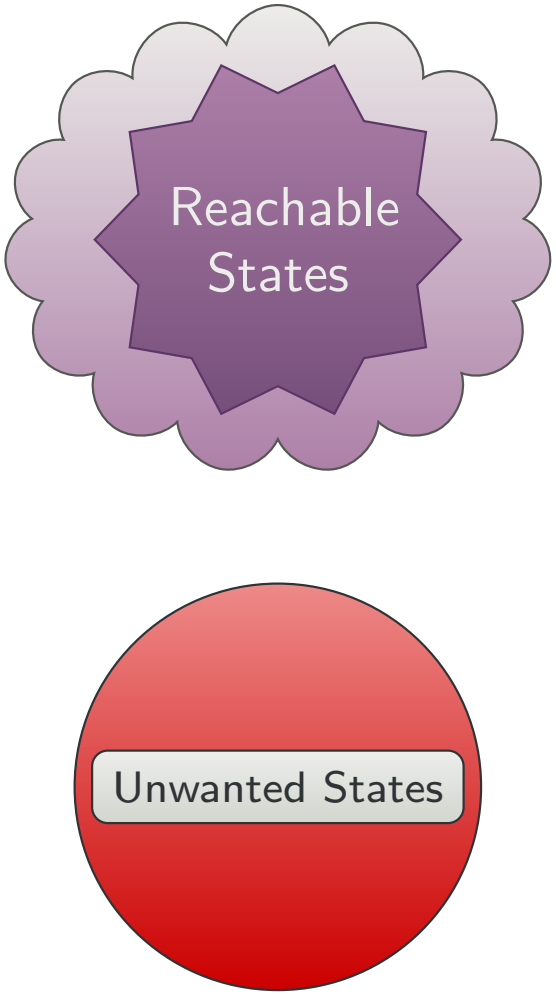
Trusting Programs: Existing Methods

Proving programs

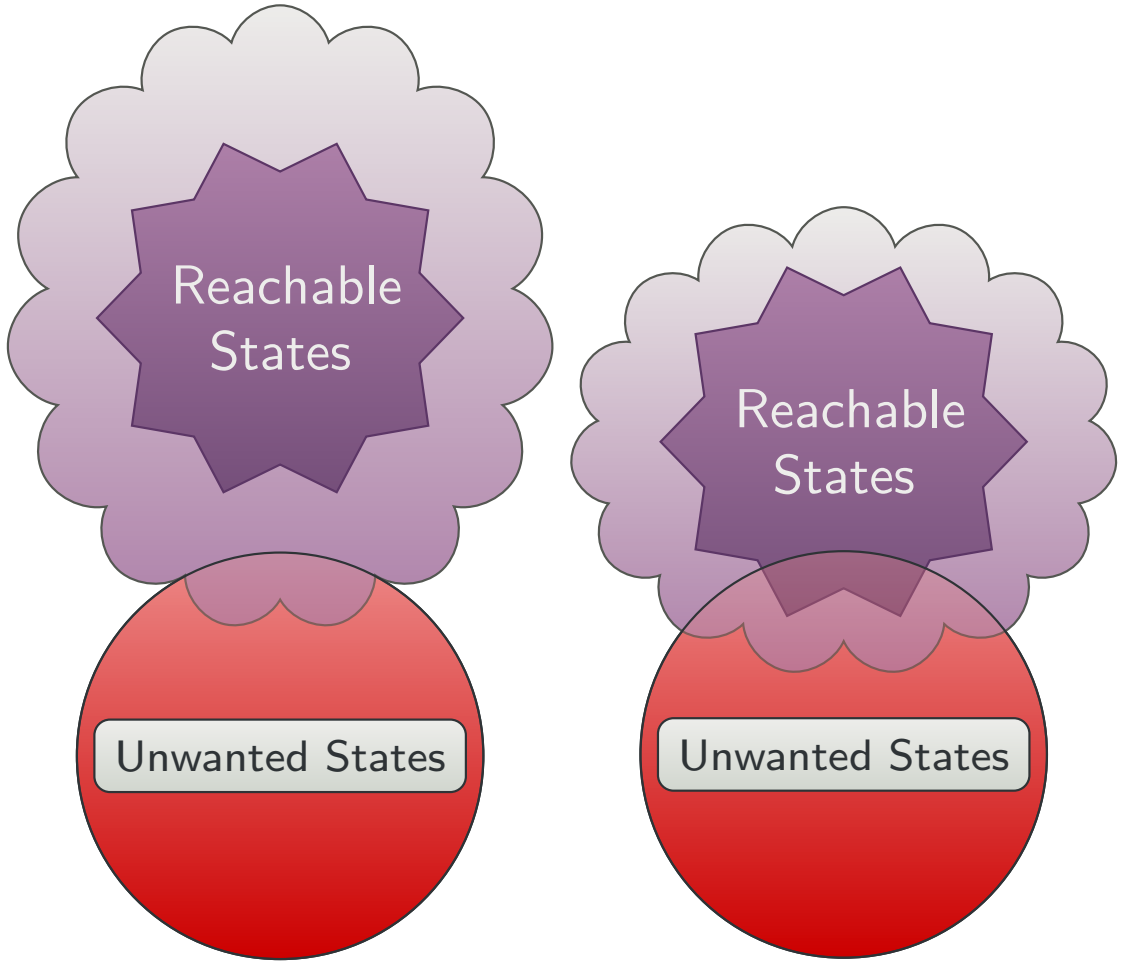
Complex, but provide strong guarantees.



Approximations



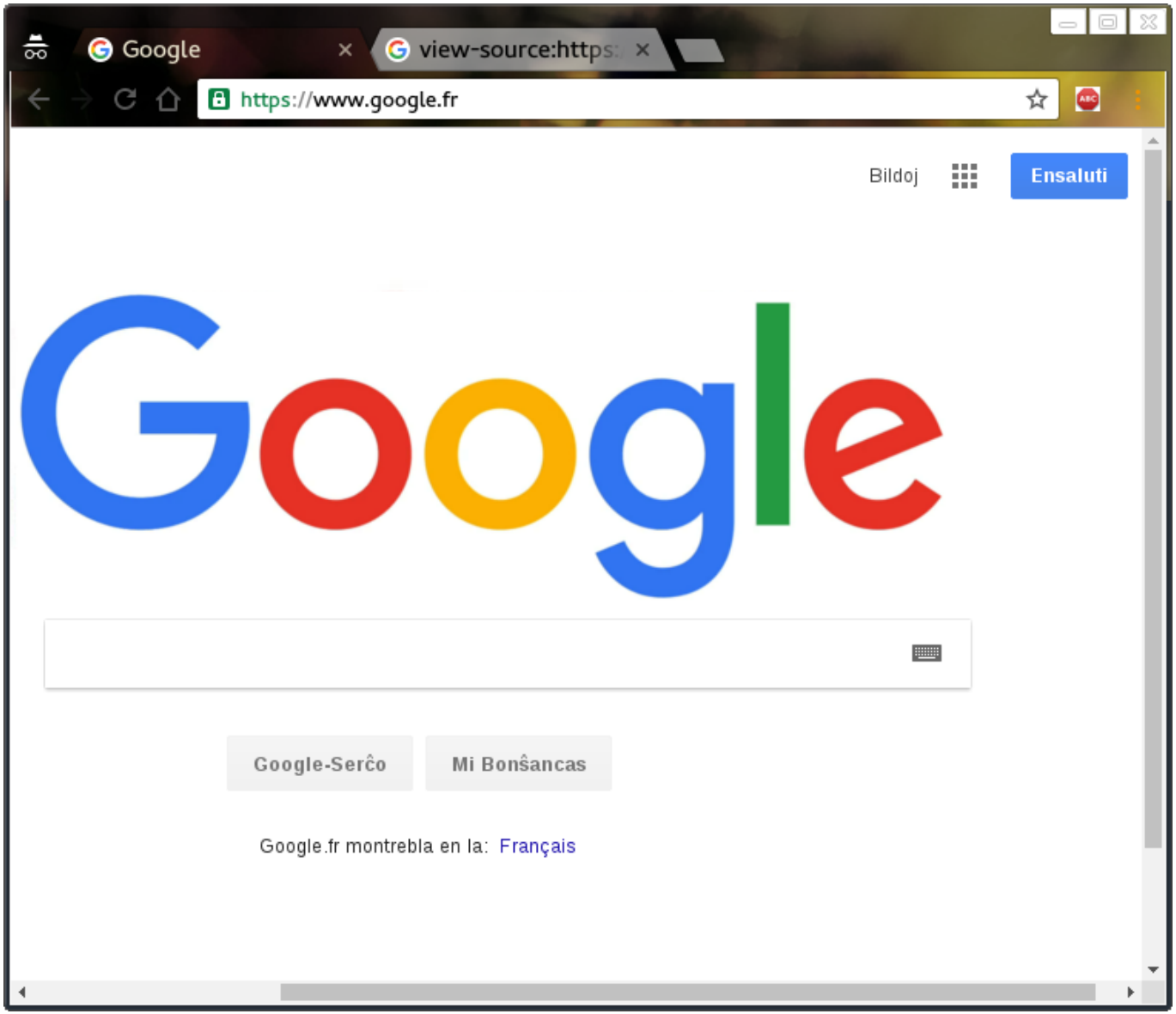
Safety proven



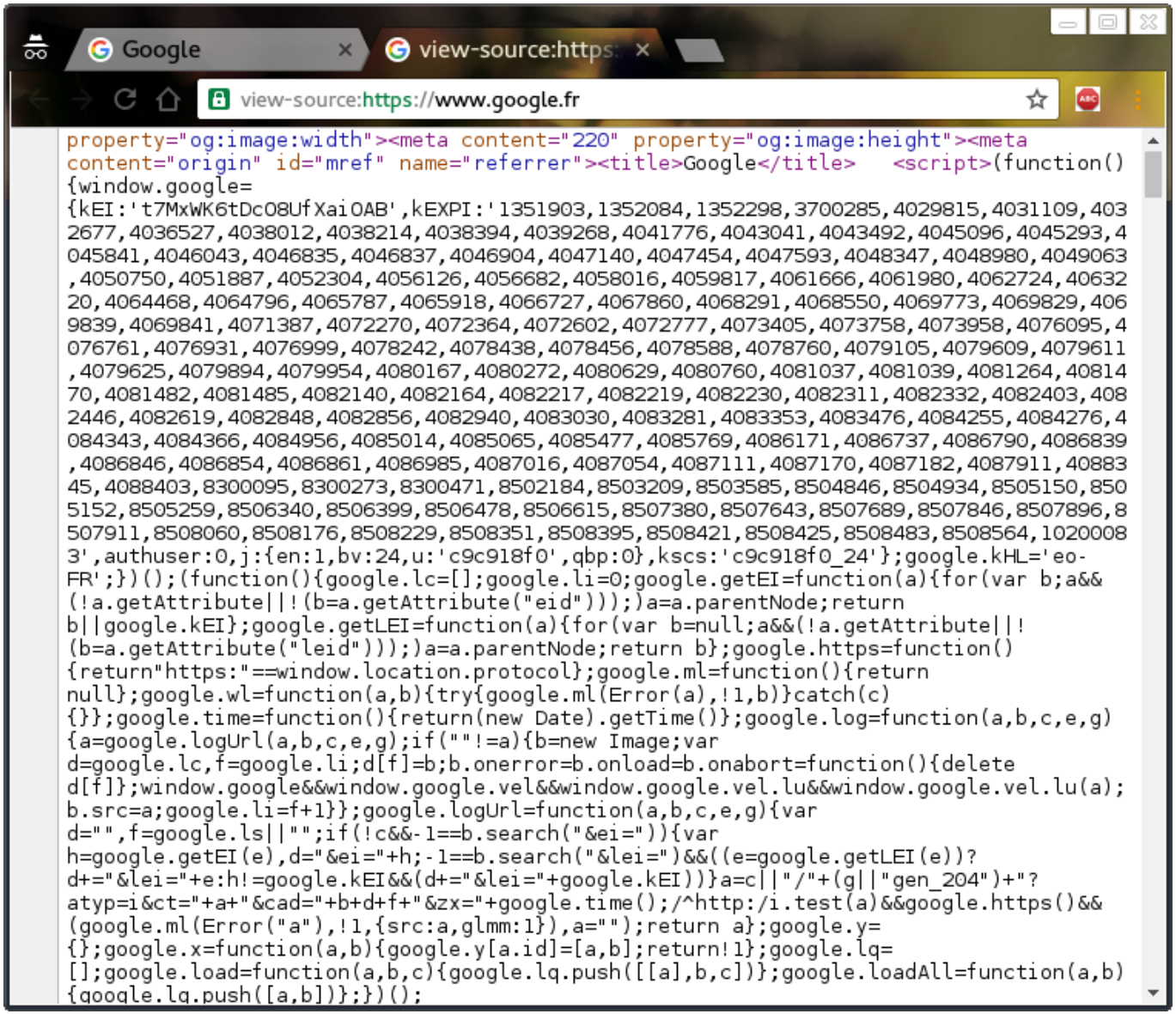
Unconclusive

JavaScript

The Language of the Web

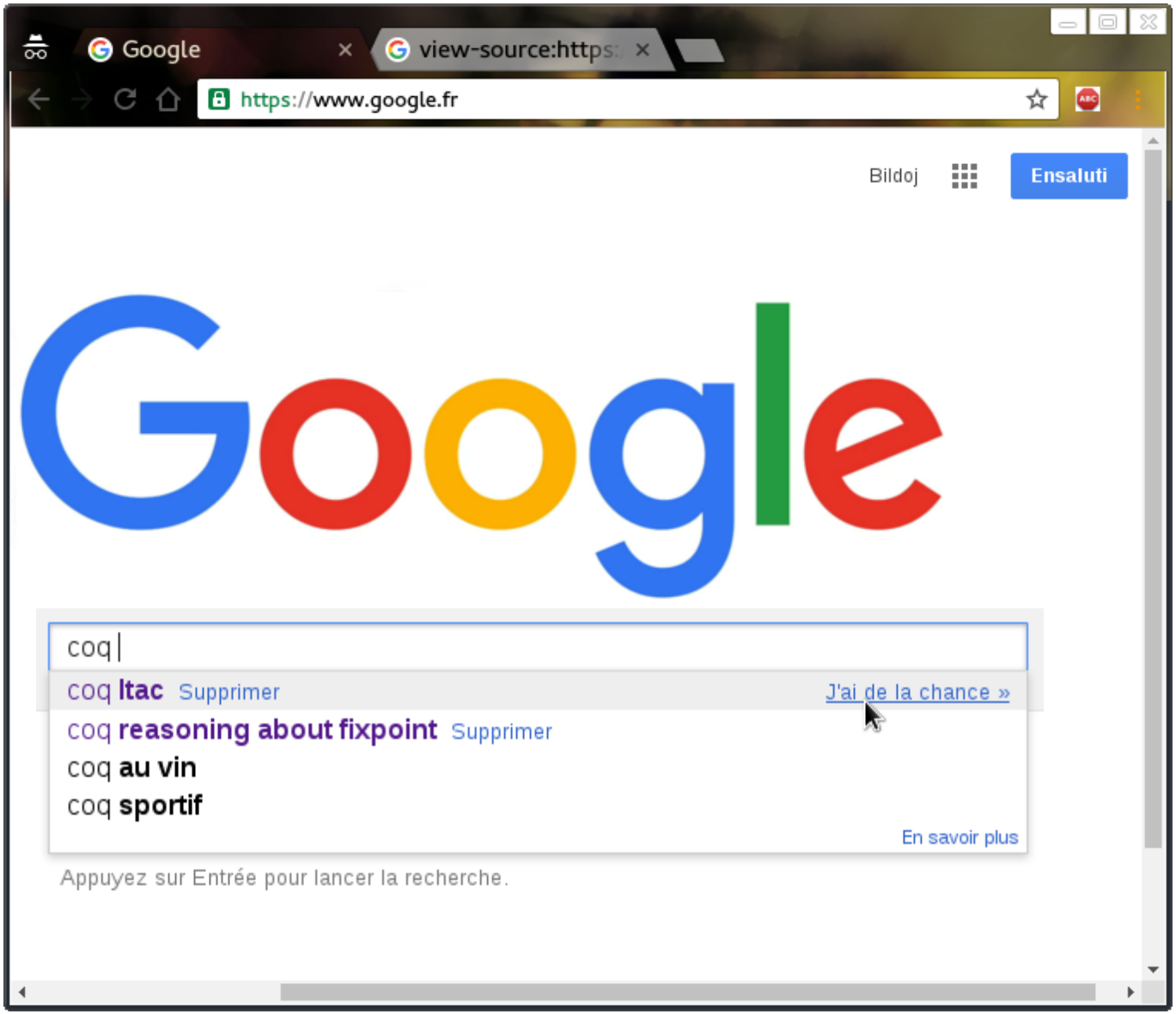


The Language of the Web

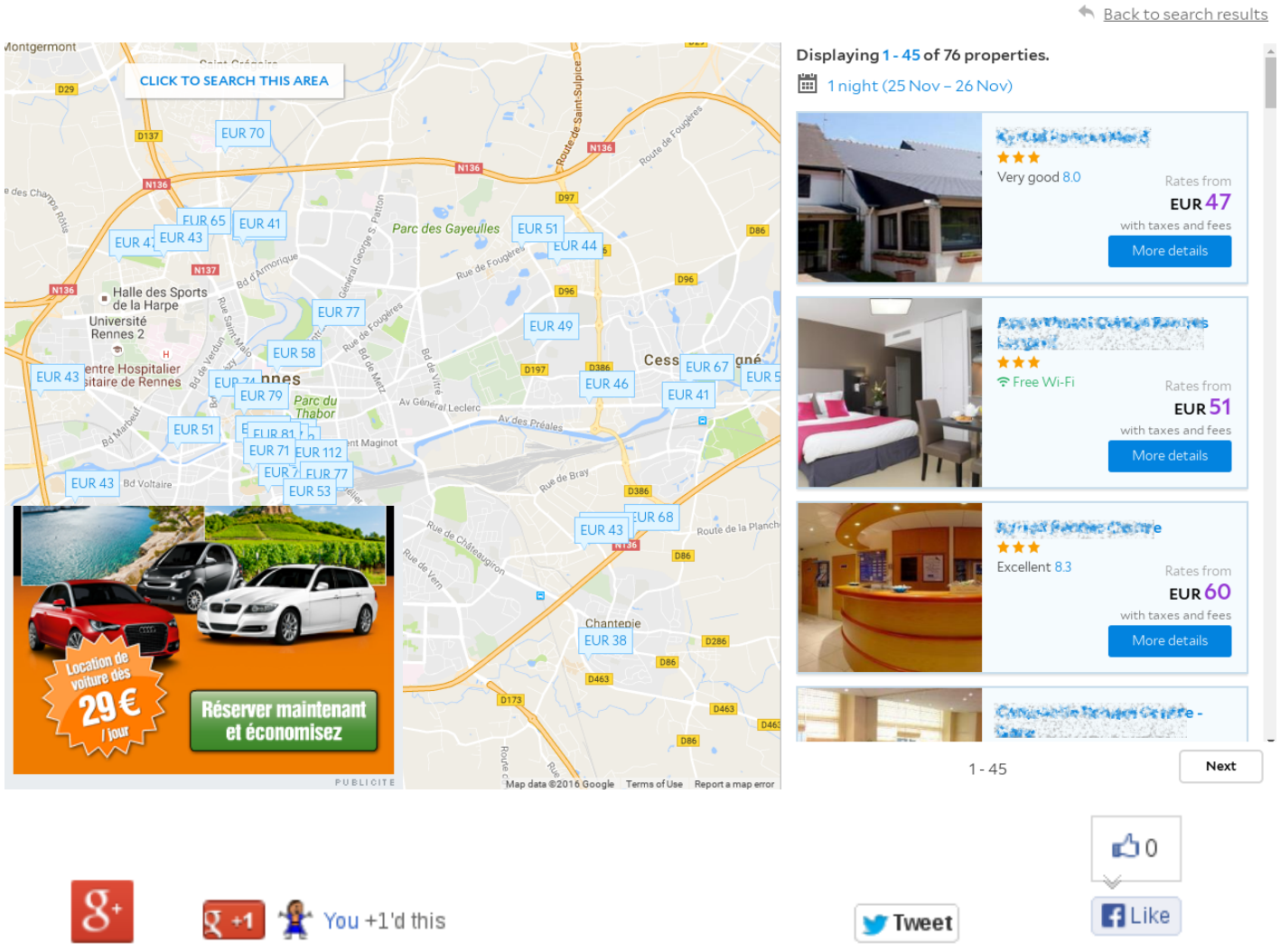


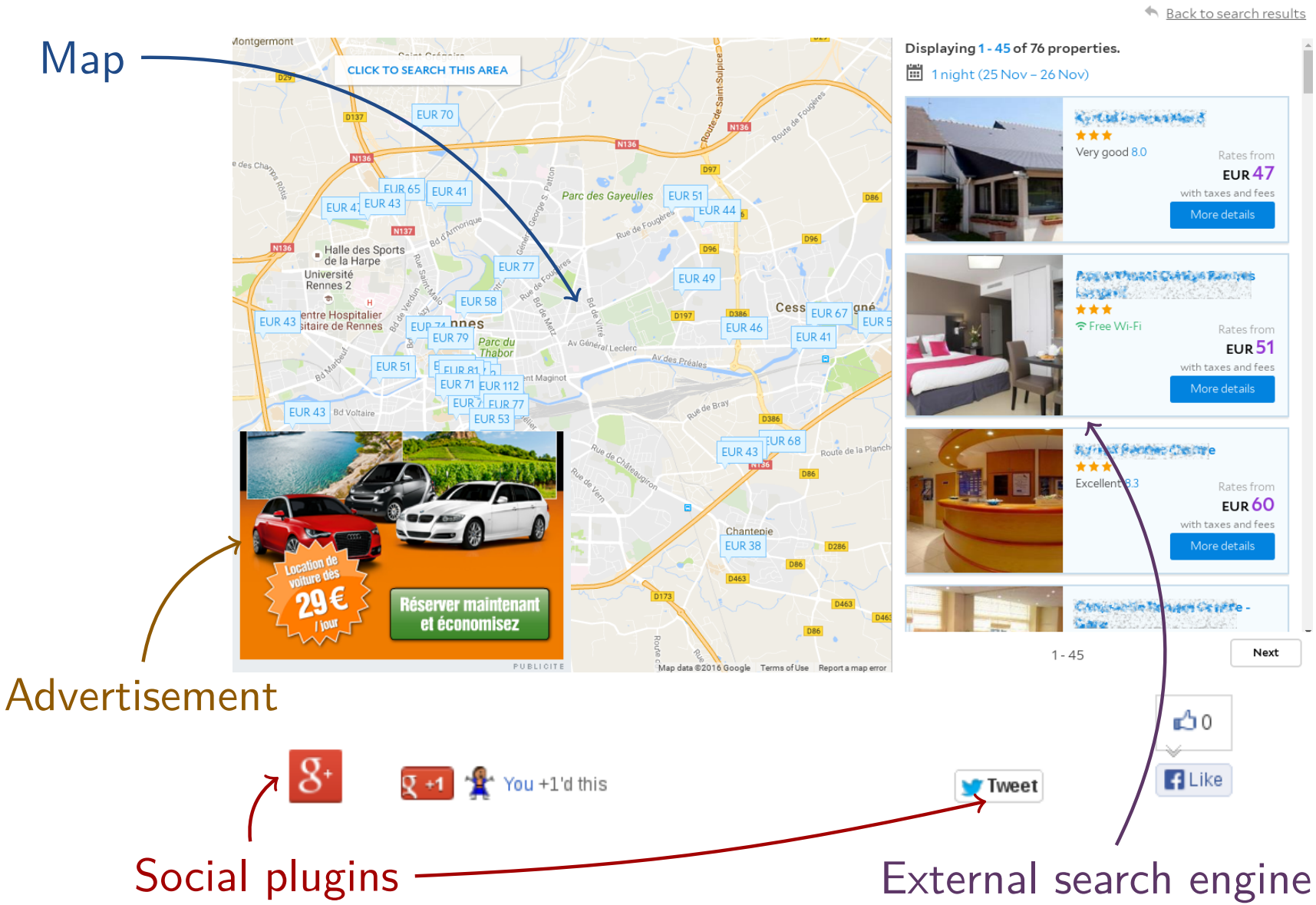
7,500 lines of JavaScript code!

The Language of the Web



7,500 lines of JavaScript code!

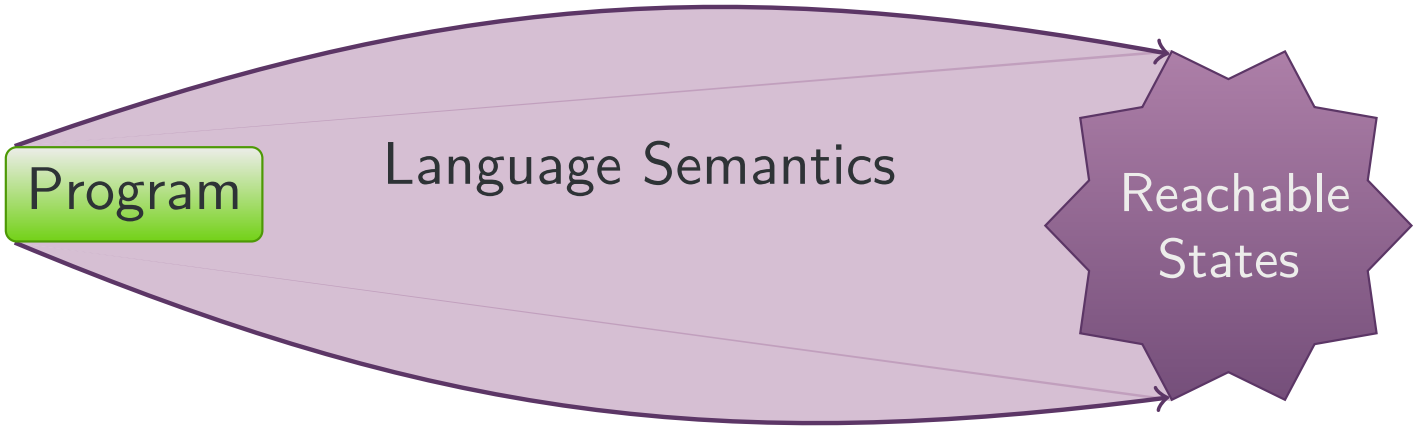

Back to search results
CLICK TO SEARCH THIS AREA
Displaying 1 - 45 of 76 properties.
1 night (25 Nov – 26 Nov)
Very good 8.0
Rates from
EUR 47
with taxes and fees
More details
Free Wi-Fi
Rates from
EUR 51
with taxes and fees
More details
Excellent 8.3
Rates from
EUR 60
with taxes and fees
More details
1 - 45
Next
Parc des Gayeulles
Parc du Thabor
Halle des Sports de la Harpe
Université Rennes 2
Chantepie
Location de voiture dès 29 € / jour
Réserver maintenant et économisez
PUBLICITE
Map data ©2016 Google
Terms of Use
Report a map error
You +1'd this
Tweet
Like
0


Map
Back to search results
CLICK TO SEARCH THIS AREA
Displaying 1 - 45 of 76 properties.
1 night (25 Nov – 26 Nov)
Very good 8.0
Rates from
EUR 47
with taxes and fees
More details
Free Wi-Fi
Rates from
EUR 51
with taxes and fees
More details
Excellent 8.3
Rates from
EUR 60
with taxes and fees
More details
1 - 45
Next
Parc des Gayeulles
Parc du Thabor
Halle des Sports de la Harpe
Université Rennes 2
Chantepie
Location de voiture dès 29 € / jour
Réserver maintenant et économisez
PUBLICITE
Map data ©2016 Google
Terms of Use
Report a map error
Advertisement
+1
You +1'd this
Tweet
0
Like
Social plugins
External search engine

JavaScript is *Specified*

This Thesis

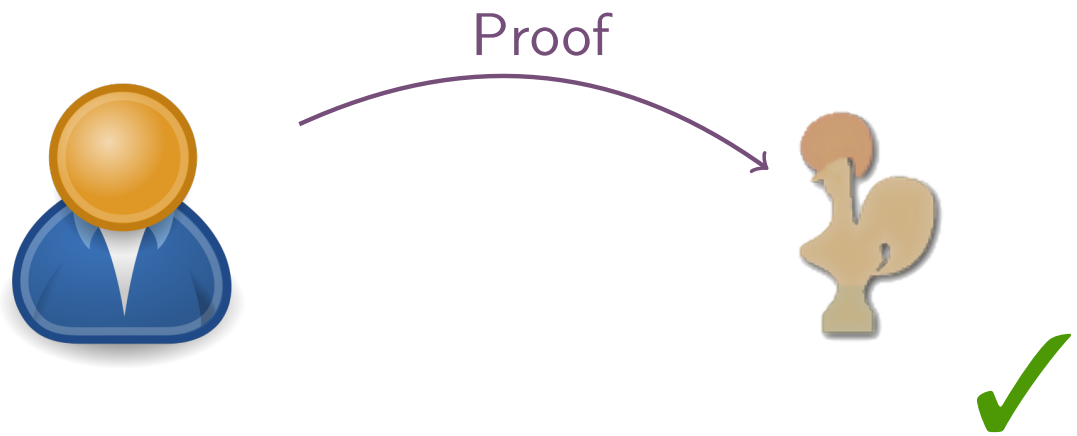
1. A Trustable Formal Semantics For JavaScript



2. Approximations of Language Semantics

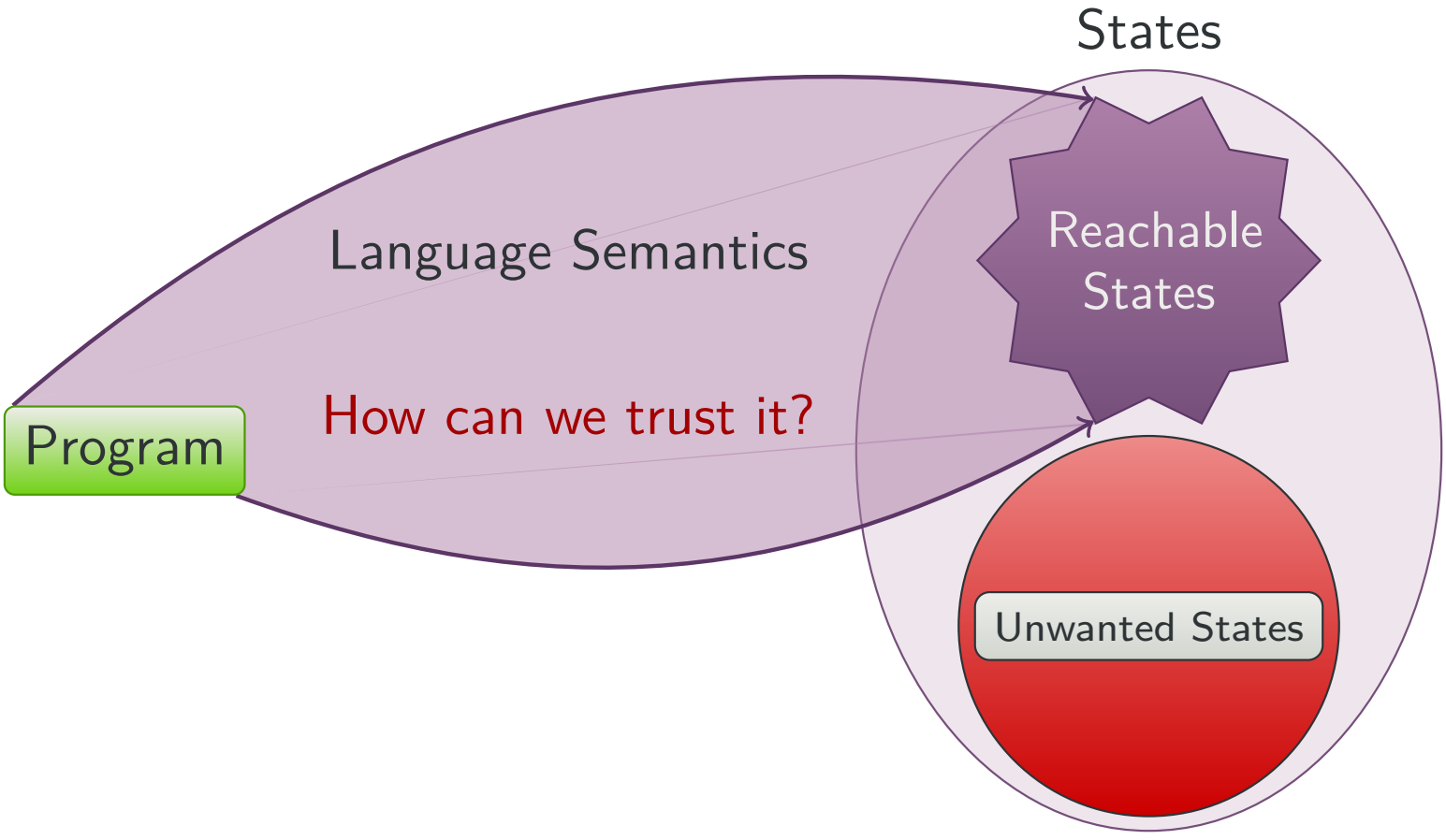
3. Elements of Analysis of JavaScript

The Coq Proof Assistant



A Trustable Formal Semantics For JavaScript

Language Semantics



Martin Bodin et al. “A Trusted Mechanised JavaScript Specification”. In: *POPL*. 2014.

Formal Semantics of JavaScript

The ECMAScript standard

ECMA International, ed. *ECMAScript Language Specification. Standard ECMA-262, Edition 5.1*. 2011.

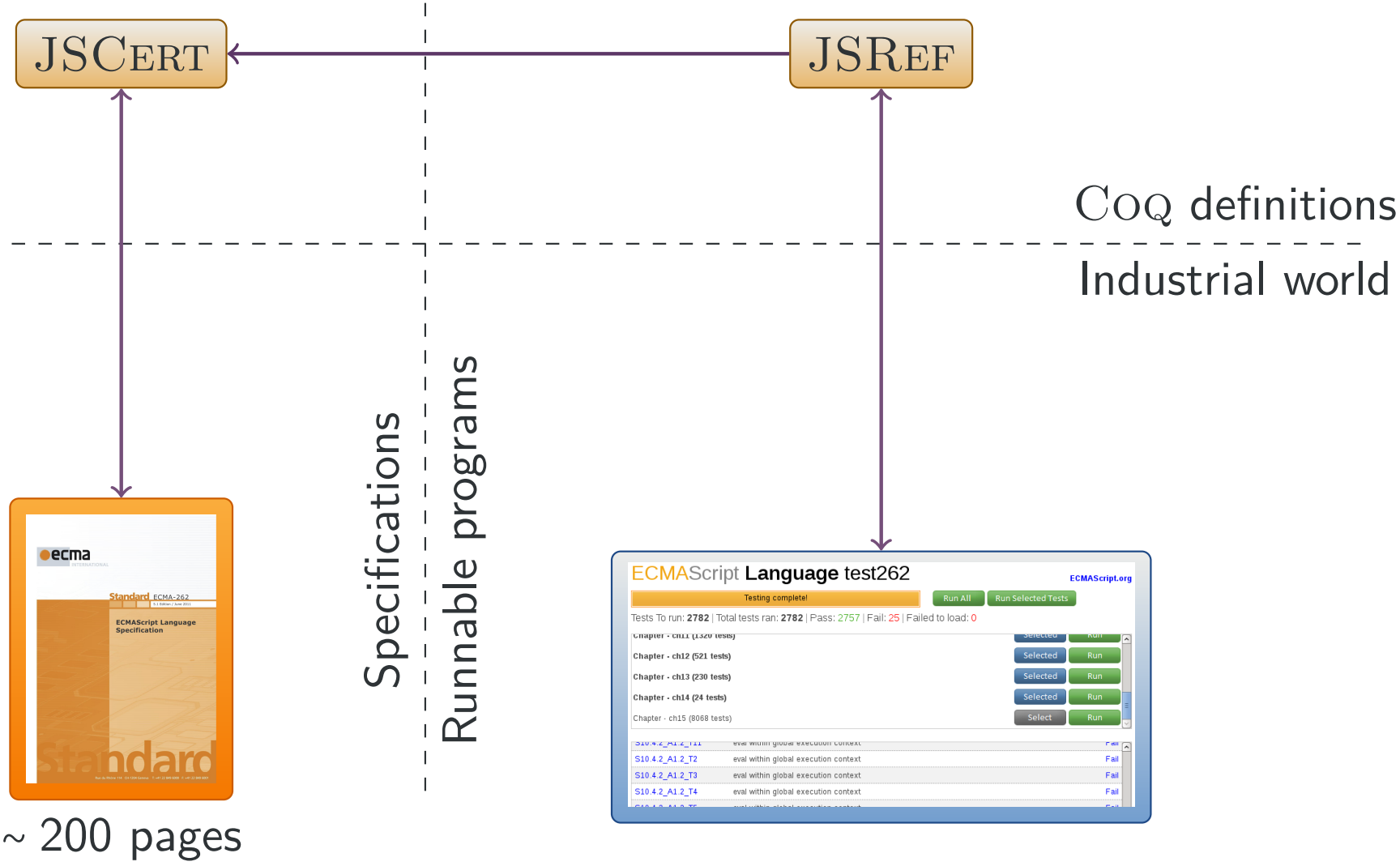
Formal Semantics Close to ECMAScript

Sergio Maffeis, John C. Mitchell, and Ankur Taly. "An Operational Semantics for JavaScript". In: *APLAS*. 2008.

Formal Semantics Executable

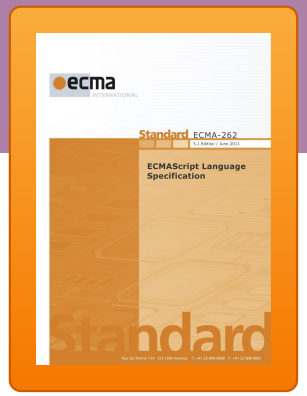
Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. "The Essence of JavaScript". In: *ECOOP*. 2010.

The JSCert Project



~ 200 pages

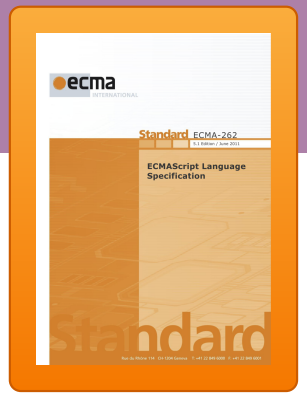
ECMAScript



"`s1 ; s2`" is evaluated as follows.

1. Let o_1 be the result of evaluating `s1`.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating `s2`.
4. If an exception V was thrown, return $(\mathit{Throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

ECMAScript



"`s1 ; s2`" is evaluated as follows.

1. Let o_1 be the result of evaluating `s1`.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating `s2`.
4. If an exception V was thrown, return $(\mathit{Throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

Conditions Evaluation of a subterm Returning a result

Pretty-Big-Step

"`s1 ; s2`" is evaluated as follows.

1. Let o_1 be the result of evaluating `s1`.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating `s2`.

Conditions Evaluation of a subterm Returning a result

Pretty-Big-Step

"s1 ; s2" is evaluated as follows.

1. Let o_1 be the result of evaluating s1.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating s2.

Conditions Evaluation of a subterm Returning a result

$$\text{SEQ-}1(s_1, s_2)\quad \frac{\sigma, s_1 \Downarrow o_1 \qquad o_1, seq_1\ s_2 \Downarrow o}{\sigma, seq\ s_1\ s_2 \Downarrow o}$$

$$\text{SEQ-}2(s_2)\quad \frac{}{o_1, seq_1\ s_2 \Downarrow o_1}\ \textbf{abort}\ o_1$$

$$\text{SEQ-}3(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, seq_2 \Downarrow o}{o_1, seq_1\ s_2 \Downarrow o}\ \neg\textbf{abort}\ o_1 \qquad \ldots$$

The JSCert Project

JSCert

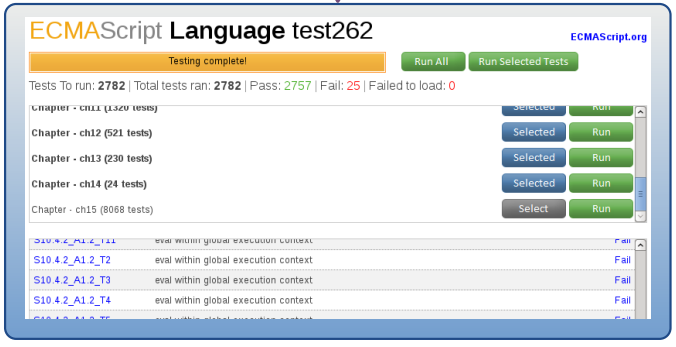
JSRef

Step / rule correspondence

Running tests

Coq definitions

Industrial world

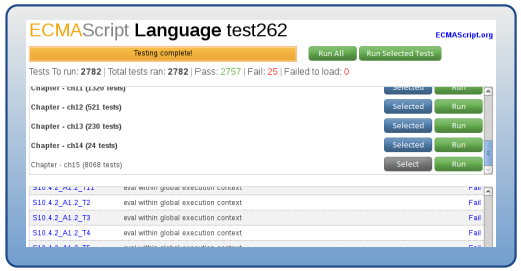


Sequence in JSRef

```
1  Definition run_seq S (s1 s2 : stat) : result :=
2    if_success (run_stat S s1) (fun S1 o1 =>
3      if_success (run_stat S1 s2) (fun S2 o2 =>
4        (* ... *))).
```

Sequence in JSRef

```
1  Definition run_seq S (s1 s2 : stat) : result :=
2    if_success (run_stat S s1) (fun S1 o1 =>
3      if_success (run_stat S1 s2) (fun S2 o2 =>
4        (* ... *))).
```



- JSRef is executable and can be tested.

```
1  while (1 === 1){
2    var v = "reached" ;
3    break
4  }
5  if (v !== "reached")
6    $ERROR ("v === 'reached'. Actual: v === " + v)
```

Correctness Theorem

```
1 Theorem run_javascript_correct : forall p o,
2   run_javascript p = Some o →
3   red_javascript p o.
```

Martin Bodin and Alan Schmitt. “A Certified JavaScript Interpreter”. In: *JFLA*. 2013.

The JSCert Project

~ 900 rules

~ 3,000 lines of code

JSCert ← Correctness → JSRef

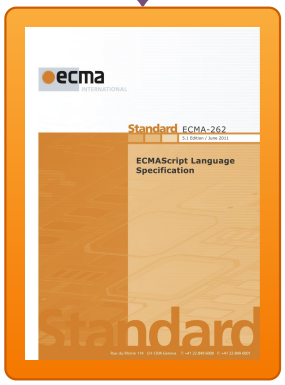
~ 6,000 lines of proof

Step / rule correspondence

Running tests

Coq definitions

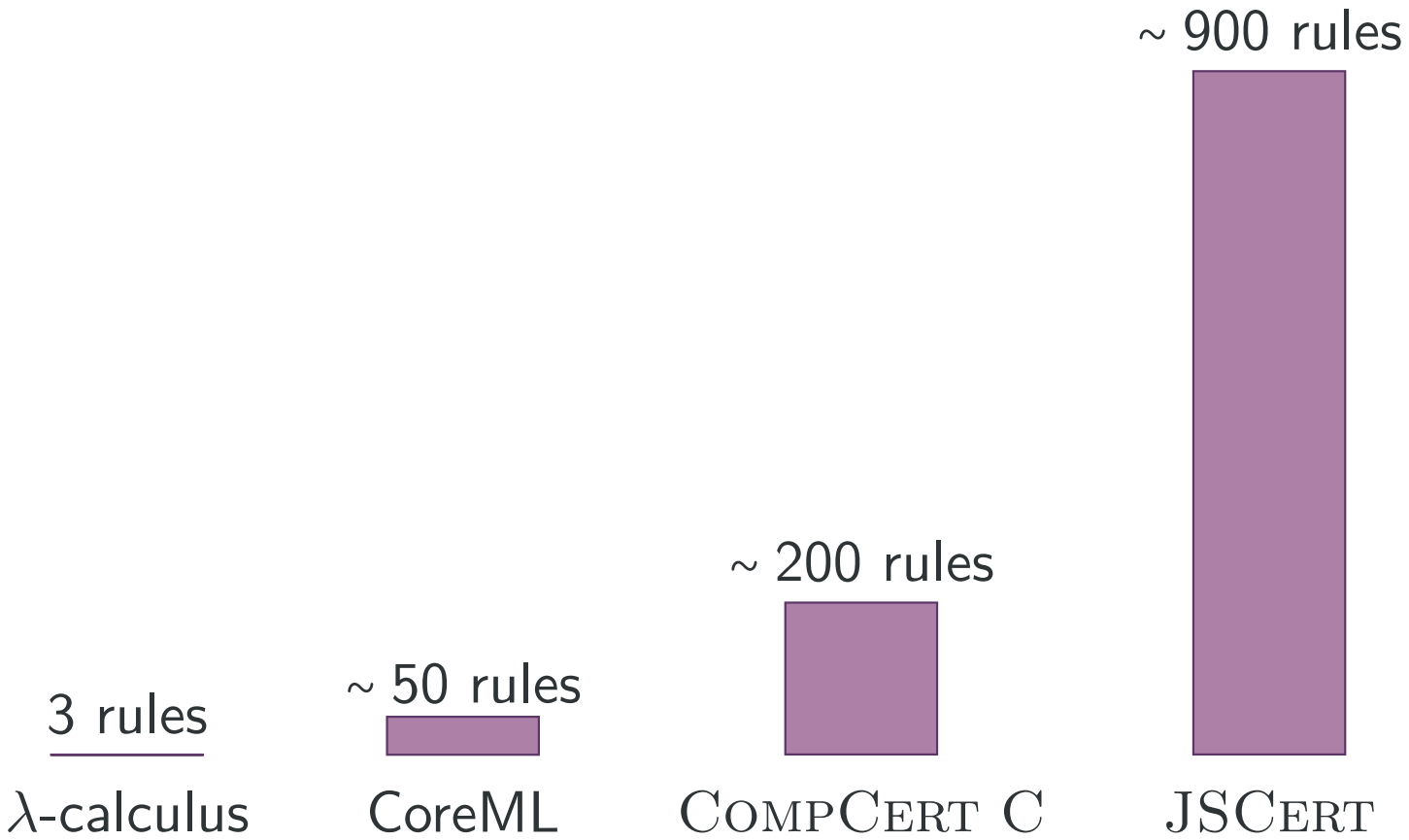
Industrial world



~ 900 steps

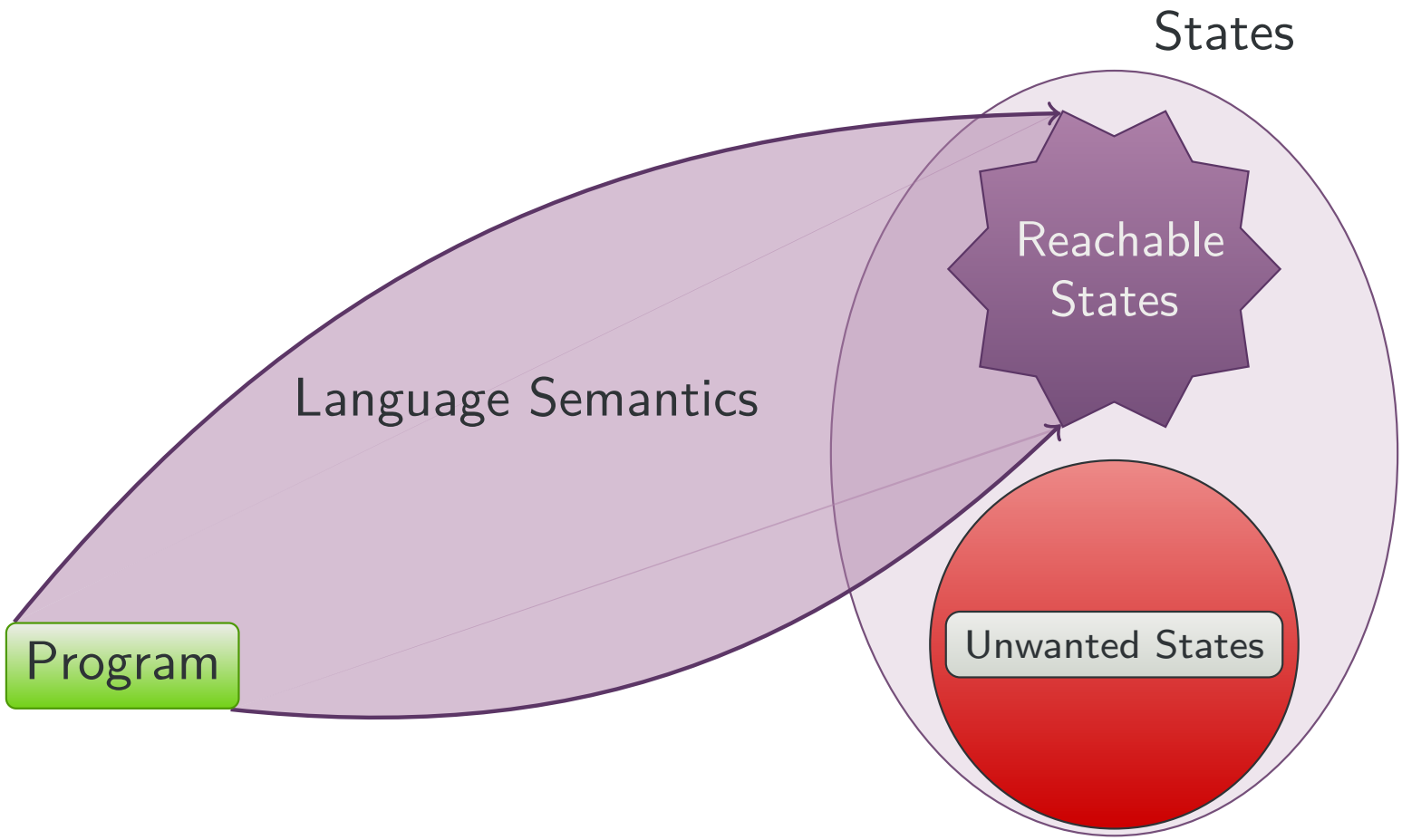
5,126 tests passed

Semantic Sizes



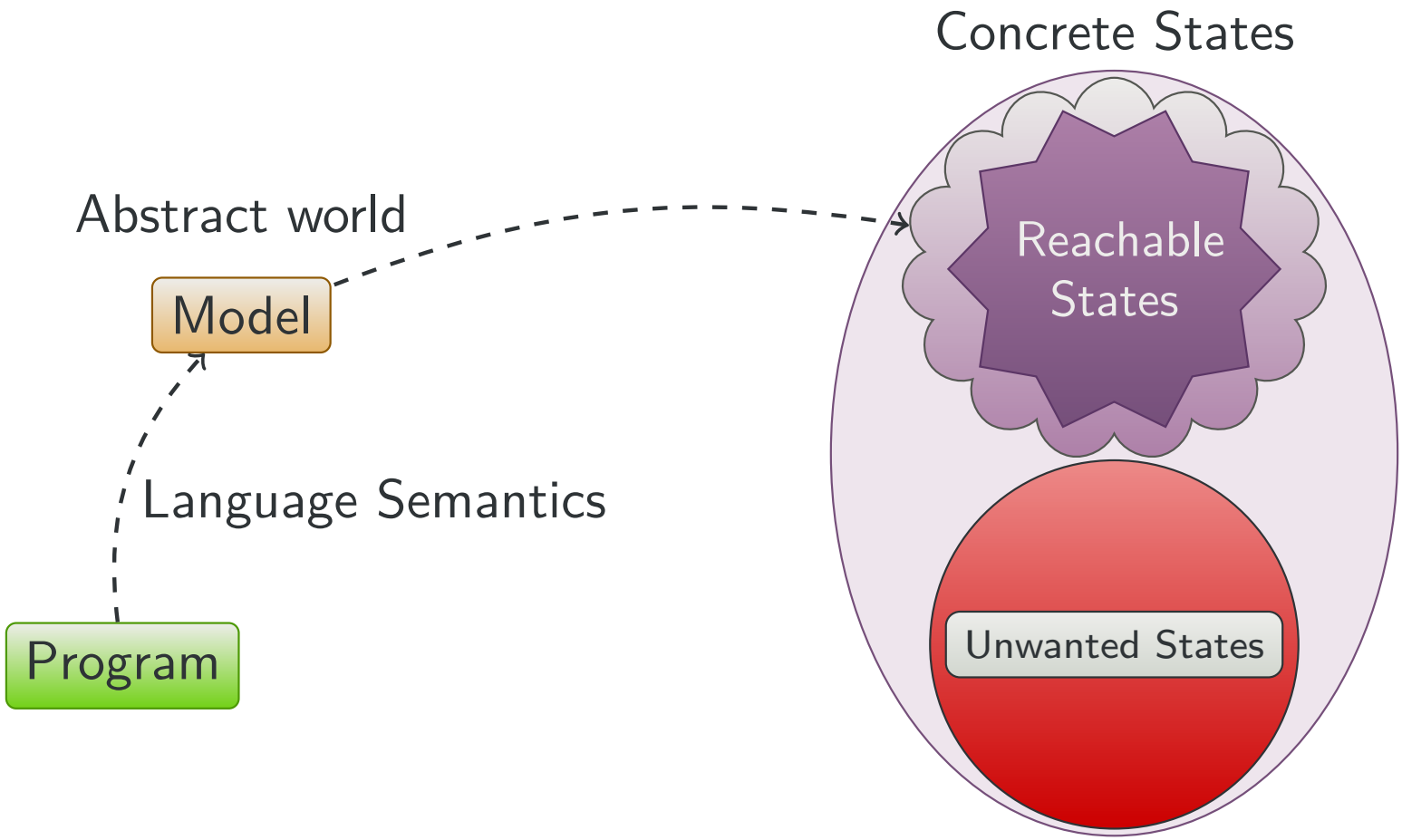
Approximations of Language Semantics

Certified Analyses



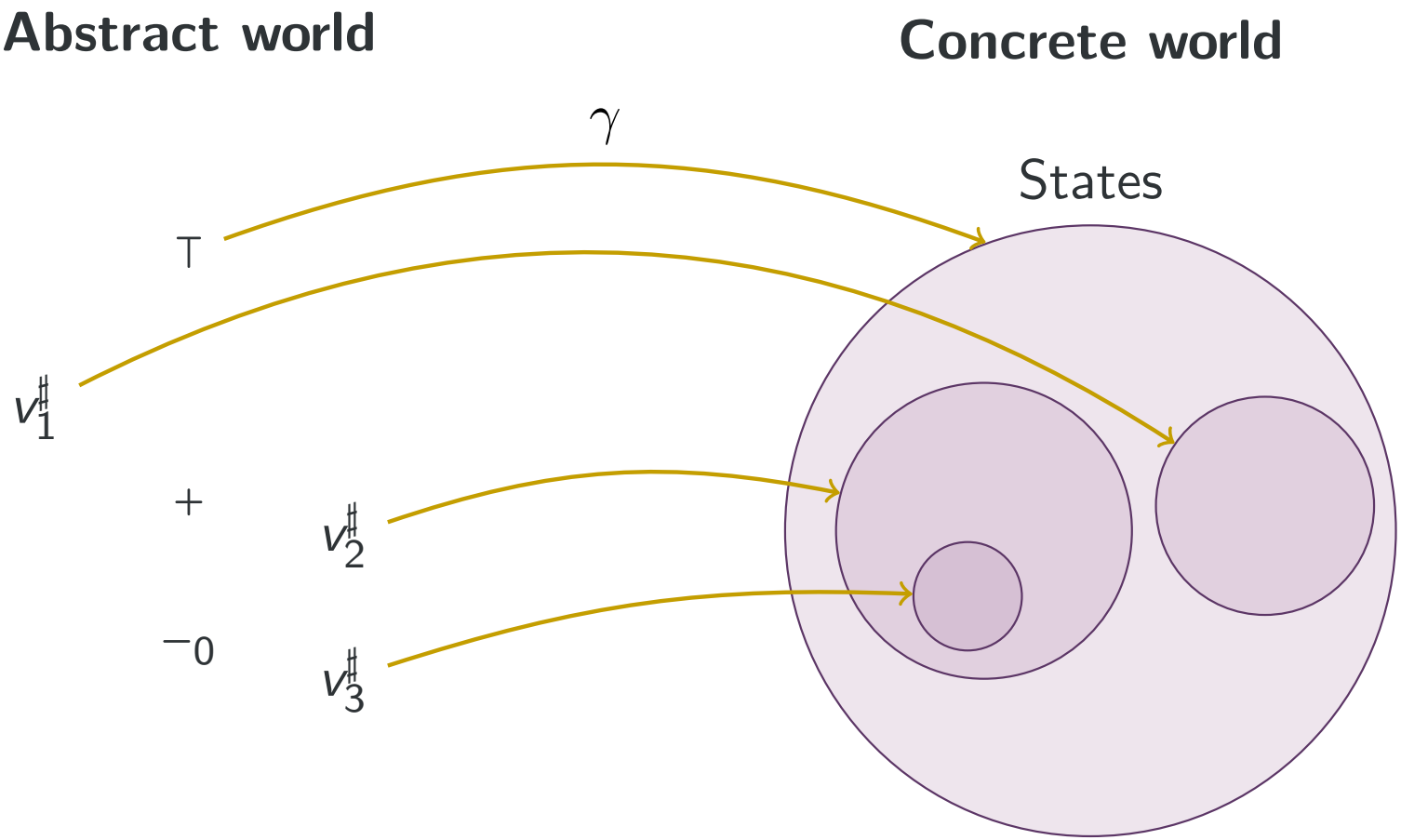
Martin Bodin, Thomas Jensen, and Alan Schmitt. "Certified Abstract Interpretation with Pretty-Big-Step Semantics". In: *CPP*. 2015.

Certified Analyses



Martin Bodin, Thomas Jensen, and Alan Schmitt. "Certified Abstract Interpretation with Pretty-Big-Step Semantics". In: *CPP*. 2015.

Abstract Interpretation



$$\gamma(+) = \mathbb{Z}_+^\star$$

$$\gamma(-_0) = \mathbb{Z}_-$$

Abstracting Derivations

A *rule based* approach

David A. Schmidt. "Natural-Semantics-Based Abstract Interpretation (preliminary version)". In: *SAS*. 1995.

$$\text{IF-POS}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star \qquad \text{IF-NEG}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$$

Abstract derivations have to *cover* concrete derivations

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \hline E^\sharp, e \Downarrow^\sharp + & \hline E^\sharp, s_1 \Downarrow^\sharp E'^\sharp\end{array}}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \text{IF-POS}(e, s_1, s_2)$$

Abstracting Derivations

A *rule based* approach

David A. Schmidt. "Natural-Semantics-Based Abstract Interpretation (preliminary version)". In: *SAS*. 1995.

$$\text{IF-POS}(e, s_1, s_2)\ \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star \qquad \text{IF-NEG}(e, s_1, s_2)\ \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$$

Abstract derivations have to *cover* concrete derivations

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \hline E^\sharp, e \Downarrow^\sharp - & \end{array} \qquad \begin{array}{c}\vdots \\ \hline E^\sharp, s_2 \Downarrow^\sharp E'^\sharp\end{array}}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \text{IF-NEG}(e, s_1, s_2)$$

Abstracting Derivations

A *rule based* approach

David A. Schmidt. "Natural-Semantics-Based Abstract Interpretation (preliminary version)". In: *SAS*. 1995.

$$\text{IF-POS}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star$$

$$\text{IF-NEG}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$$

Abstract derivations have to *cover* concrete derivations

$$\frac{\dfrac{\vdots}{E^\sharp, e \Downarrow^\sharp \top}}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp}$$

Abstracting Derivations

A *rule based* approach

David A. Schmidt. "Natural-Semantics-Based Abstract Interpretation (preliminary version)". In: *SAS*. 1995.

$$\textsc{if-pos}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star \qquad \textsc{if-neg}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$$

Abstract derivations have to *cover* concrete derivations

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \hline E^\sharp, e \Downarrow^\sharp \top & \hline E^\sharp, s_1 \Downarrow^\sharp E_1^\sharp\end{array}}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E_1^\sharp}$$

Abstracting Derivations

A *rule based* approach

David A. Schmidt. "Natural-Semantics-Based Abstract Interpretation (preliminary version)". In: *SAS*. 1995.

$$\textsc{if-pos}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star$$

$$\textsc{if-neg}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$$

Abstract derivations have to *cover* concrete derivations

$$\frac{\vdots \qquad \vdots \qquad \vdots}{E^\sharp, e \Downarrow^\sharp \top \qquad E^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp E_2^\sharp}$$
$$E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp$$

Local Requirements

Our motto

one concrete rule = one abstract rule

Concrete rules

$$\text{IF-POS}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\quad v \in \mathbb{Z}_+^\star$$

$$\text{IF-NEG}(e, s_1, s_2)\quad \frac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow E'}\quad v \in \mathbb{Z}_-$$

Abstract rules

$$\text{IF-POS}(e, s_1, s_2)\quad \frac{E^\sharp, e \Downarrow^\sharp v^\sharp \qquad E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\quad \gamma\left(v^\sharp\right) \cap \mathbb{Z}_+^\star \neq \varnothing$$

$$\text{IF-NEG}(e, s_1, s_2)\quad \frac{E^\sharp, e \Downarrow^\sharp v^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\quad \gamma\left(v^\sharp\right) \cap \mathbb{Z}_- \neq \varnothing$$

Local Requirements

Our motto

one concrete rule = one abstract rule

We only require *local* properties to be proven.

Concrete rules	**Abstract rules**
IF-POS(e, s_1, s_2)	IF-POS(e, s_1, s_2)
$\dfrac{E, e \Downarrow v \qquad E, s_1 \Downarrow E'}{E, \mathit{if}\,(e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_+^\star$	$\dfrac{E^\sharp, e \Downarrow^\sharp v^\sharp \qquad E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\,(e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \gamma\left(v^\sharp\right) \cap \mathbb{Z}_+^\star \neq \varnothing$
IF-NEG(e, s_1, s_2)	IF-NEG(e, s_1, s_2)
$\dfrac{E, e \Downarrow v \qquad E, s_2 \Downarrow E'}{E, \mathit{if}\,(e > 0)\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}_-$	$\dfrac{E^\sharp, e \Downarrow^\sharp v^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\,(e > 0)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \gamma\left(v^\sharp\right) \cap \mathbb{Z}_- \neq \varnothing$

Abstract Semantics

- The abstract semantics $\Downarrow^\sharp$ is defined from the abstract rules.
- At each step, it applies *all* application rules.

$$\dfrac{E_0^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \qquad E_0^\sharp, s_2 \Downarrow^\sharp E_2^\sharp}{E_0^\sharp, \top, \mathit{if}\ s_1\ s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}\ \uparrow \text{IF-POS-1}(s_1, s_2)\ \uparrow \text{IF-NEG-1}(s_1, s_2)$$

Metatheorem (Soundness)

Whatever the language semantics, the abstract derivations build by $\Downarrow^\sharp$ cover *all* the corresponding concrete derivations.

Abstract Semantics

- The abstract semantics $\Downarrow^\sharp$ is defined from the abstract rules.
- At each step, it applies *all* application rules.

$$\dfrac{E_0^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \qquad E_0^\sharp, s_2 \Downarrow^\sharp E_2^\sharp}{E_0^\sharp, \top, \mathit{if}\ s_1\ s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}\ \begin{array}{l}\uparrow \textsc{if-pos-1}(s_1, s_2) \\ \uparrow \textsc{if-neg-1}(s_1, s_2)\end{array}$$

Metatheorem (Soundness)

Let p be a program, σ and $\sigma^\sharp$ a concrete and an abstract semantic contexts, and r and $r^\sharp$ a concrete and an abstract results.

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

Proving With Pretty-Big-Step

1. Let o_1 be the result of evaluating `s1`.
2. If o_1 is an exception, return o_1.

Conditions Evaluation of a subterm Returning a result

$$\frac{}{\sigma, \mathfrak{l}_{\mathfrak{r}} \Downarrow ax(\sigma)}\ cond_{\mathfrak{r}}(\sigma)$$

$$\frac{up(\sigma), \mathfrak{u}_{1,\mathfrak{r}} \Downarrow r}{\sigma, \mathfrak{l}_{\mathfrak{r}} \Downarrow r}\ cond_{\mathfrak{r}}(\sigma)$$

$$\frac{up(\sigma), \mathfrak{u}_{2,\mathfrak{r}} \Downarrow r \qquad next(\sigma, r), \mathfrak{u}_{2,\mathfrak{r}} \Downarrow r'}{\sigma, \mathfrak{l}_{\mathfrak{r}} \Downarrow r'}\ cond_{\mathfrak{r}}(\sigma)$$

- Rules are now Coq records.
- Only three cases in the proof.

Infinite Abstract Derivations

$$\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^{\sharp} \{x \mapsto \top\}$$

Infinite Abstract Derivations

$$\dfrac{\dfrac{}{\{x \mapsto \top\}, x \Downarrow^\sharp \top}\ \textsc{red-var}(x) \quad \vdots \quad \{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while}(x, x\text{--})$$

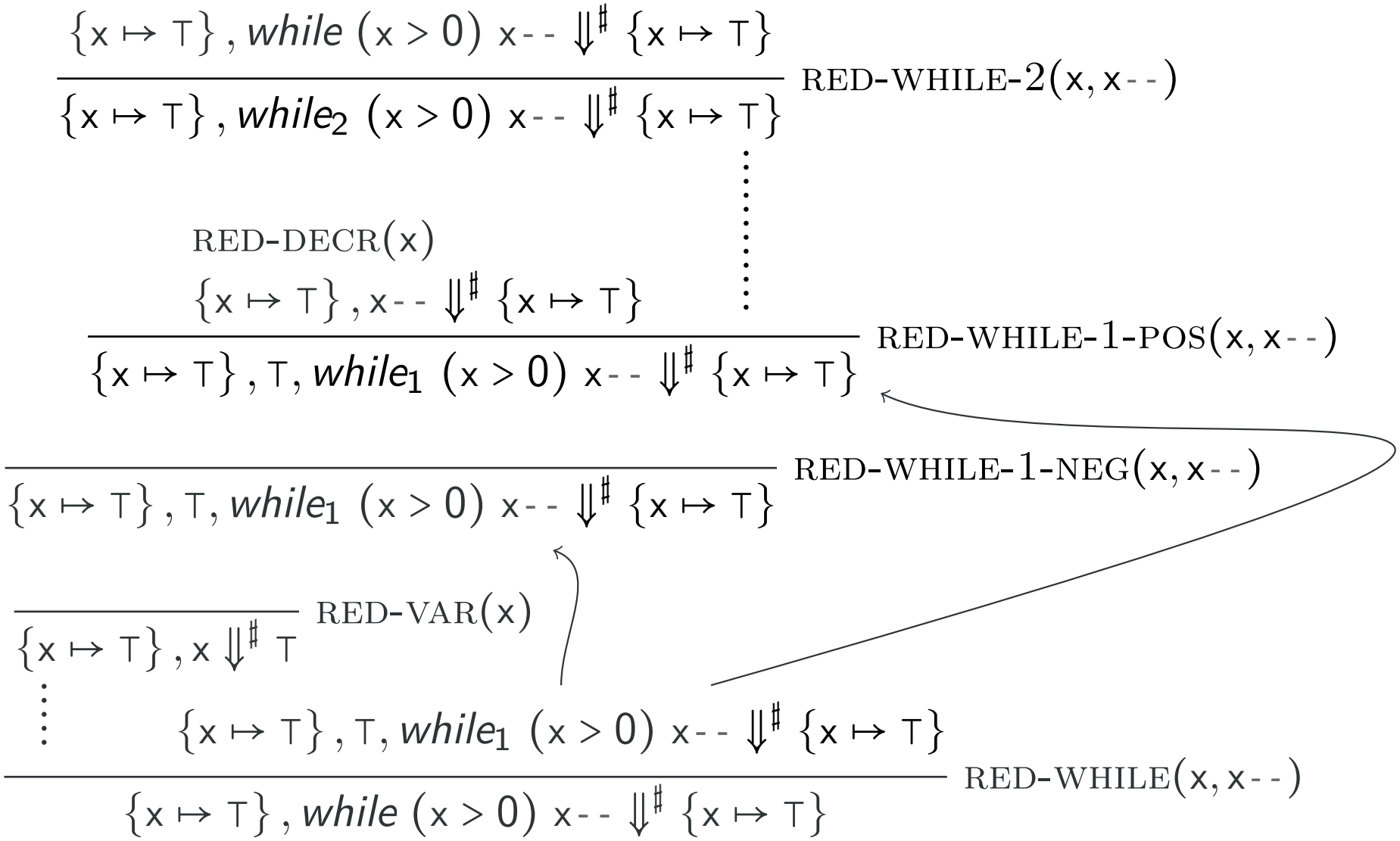
Infinite Abstract Derivations

$$\dfrac{\dfrac{}{\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while-1-neg}(x, x\text{--})}{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while}(x, x\text{--})$$

$$\dfrac{\dfrac{}{\{x \mapsto \top\}, x \Downarrow^\sharp \top}\ \textsc{red-var}(x)}{\vdots}$$

$$\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}$$

Infinite Abstract Derivations



Infinite Abstract Derivations

$$\dfrac{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}{\{x \mapsto \top\}, \mathit{while}_2\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while-2}(x, x\text{--})$$

$$\dfrac{\dfrac{\{x \mapsto \top\}, x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}{\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-decr}(x)}{\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while-1-pos}(x, x\text{--})$$

$$\dfrac{\dfrac{\dfrac{}{\{x \mapsto \top\}, x \Downarrow^\sharp \top}\ \textsc{red-var}(x)}{\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while-1-neg}(x, x\text{--})}{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while}(x, x\text{--})$$

Infinite Abstract Derivations

$$\dfrac{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}{\{x \mapsto \top\}, \mathit{while}_2\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while-2}(x, x\text{--})$$

$$\vdots$$

$$\textsc{red-decr}(x)$$

Infinite derivations subsum invariants

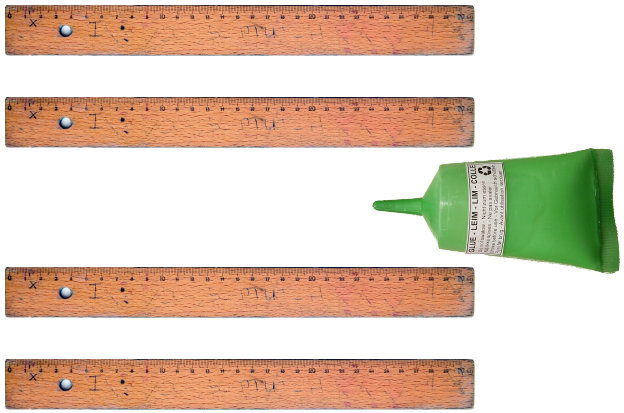
$$\textsc{while-invariant}\ \dfrac{E^\sharp, s \Downarrow^\sharp E^\sharp}{E^\sharp, \mathit{while}\ (e)\ s \Downarrow^\sharp E^\sharp}$$

$$\dfrac{}{\{x \mapsto \top\}, x \Downarrow^\sharp \top}\ \textsc{red-var}(x)$$

$$\vdots$$

$$\dfrac{\{x \mapsto \top\}, \top, \mathit{while}_1\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}{\{x \mapsto \top\}, \mathit{while}\ (x > 0)\ x\text{--} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{red-while}(x, x\text{--})$$

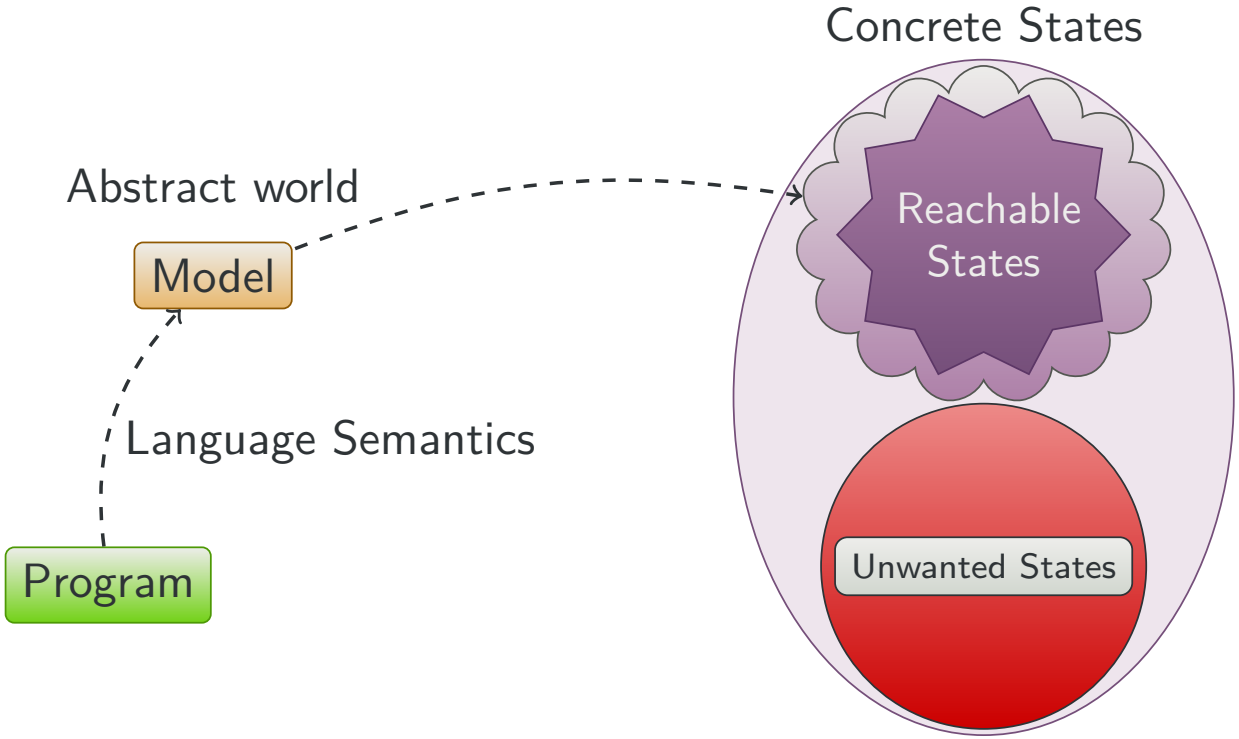
Assembling Reductions



$$\textsc{glue-weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, p \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, p \Downarrow^\sharp r^\sharp}$$

Fit into the formalism, but technical.

Building Approximations



With only *local* requirements

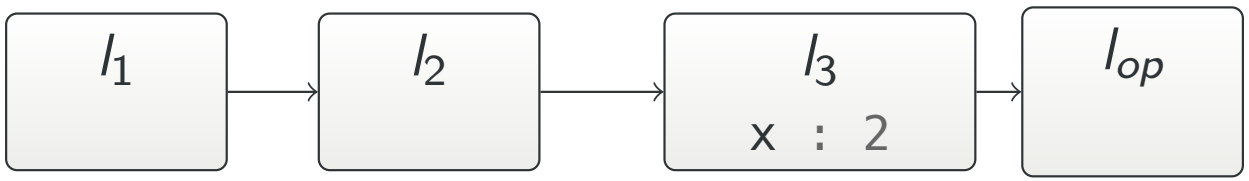
Metatheorem (Soundness)

Let p be a program, σ and $\sigma^\sharp$ a concrete and an abstract semantic contexts, and r and $r^\sharp$ a concrete and an abstract results.

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

Elements of Analysis of JavaScript

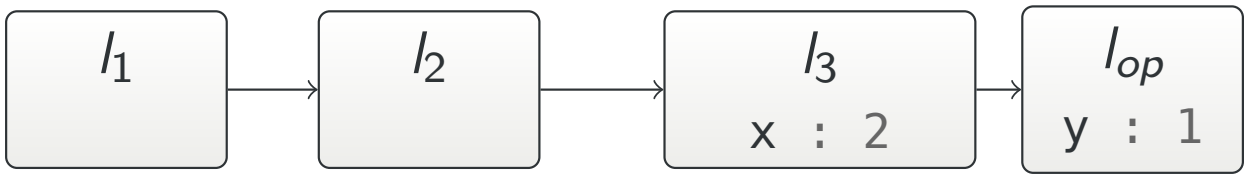
JavaScript Memory Model



JavaScript is dynamic

- Prototype inheritance;

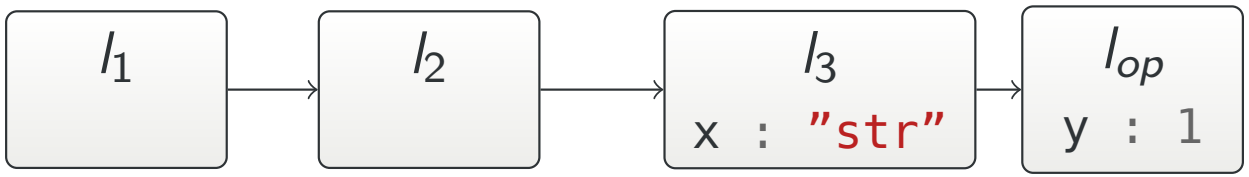
JavaScript Memory Model



JavaScript is dynamic

- Prototype inheritance;
- Dynamically adding or removing fields;
- Detecting the presence of fields;

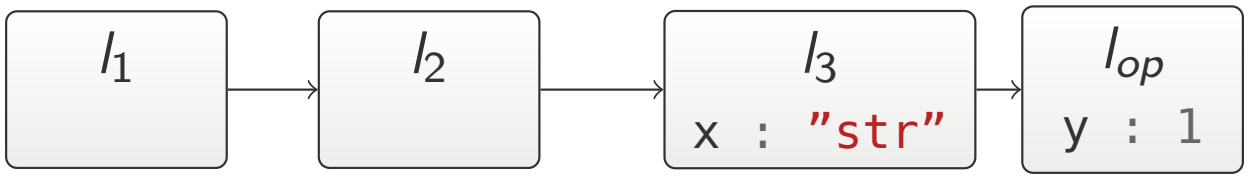
JavaScript Memory Model



JavaScript is dynamic

- Prototype inheritance;
- Dynamically adding or removing fields;
- Detecting the presence of fields;
- Changing the type of fields.

JavaScript Memory Model



JavaScript is dynamic

- Prototype inheritance;
- Dynamically adding or removing fields;
- Detecting the presence of fields;
- Changing the type of fields.

What we need to catch

- Track which fields are present;
- Mixing different types of values in the same fields.

Already a Lot of Work

Arlen Cox, Bor-Yuh Evan Chang, and Xavier Rival. "Automatic Analysis of Open Objects in Dynamic Language Programs". In: *SAS*. 2014.

Simon Holm Jensen, Anders Møller, and Peter Thiemann. "Type Analysis for JavaScript". In: *SAS*. 2009.

We aim Coq's certification

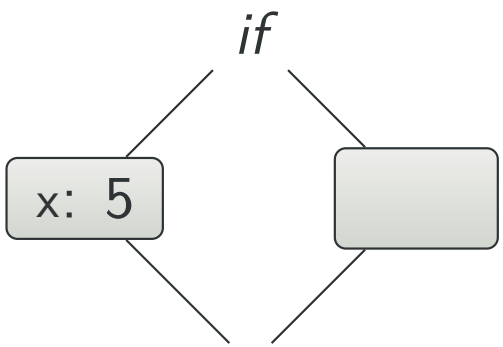
Simpler domains, but in our Coq framework.

Abstracting Objects

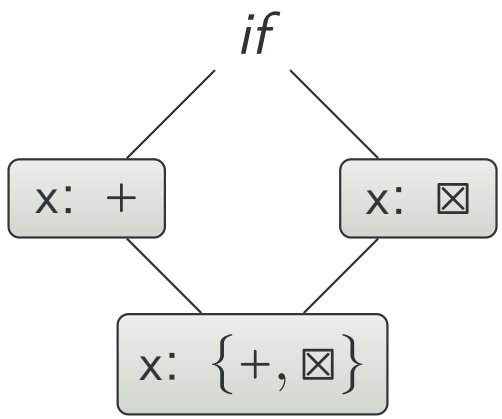
What we need to catch

- Track which fields are present;
- Mixing different types of values in the same fields.

Concrete world



Abstract world



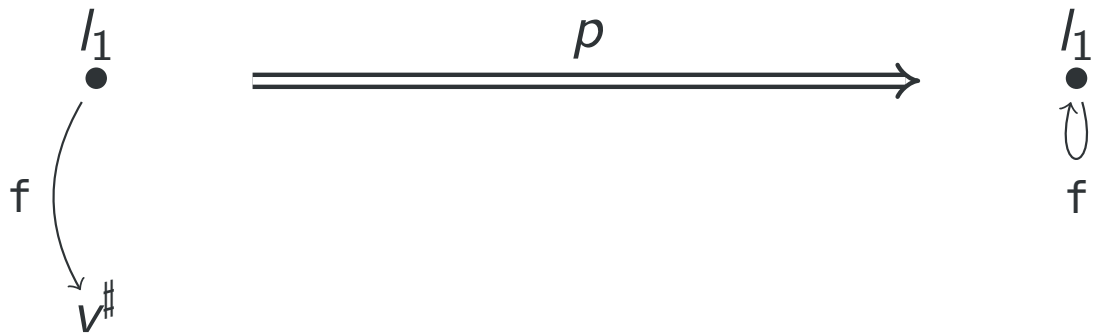
Separation Logic

$$\phi ::= emp \mid \phi_1 \star \phi_2 \mid l \mapsto o^\sharp$$

$$l_1 \mapsto \{\texttt{next} : l_2\} \star l_2 \mapsto \{\texttt{value} : v^\sharp\}$$

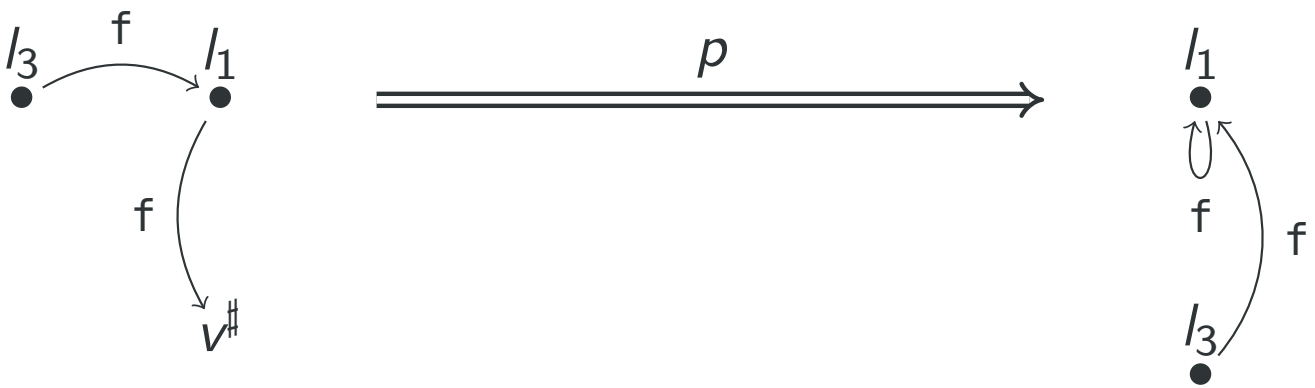
The Frame Rule

$$\textsc{glue-frame}\quad \frac{\phi, p \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, p \Downarrow^{\sharp} \phi' \star \phi_c}$$



The Frame Rule

$$\textsc{glue-frame}\quad \frac{\phi, p \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, p \Downarrow^{\sharp} \phi' \star \phi_c}$$



The Frame Rule

$$\textsc{glue-frame}\ \frac{\phi, p \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, p \Downarrow^{\sharp} \phi' \star \phi_c}$$

$$\frac{}{l_1 \mapsto \{\mathsf{f} : l_1\}, skip \Downarrow^{\sharp} l_1 \mapsto \{\mathsf{f} : l_1\}}\ \textsc{red-skip}$$

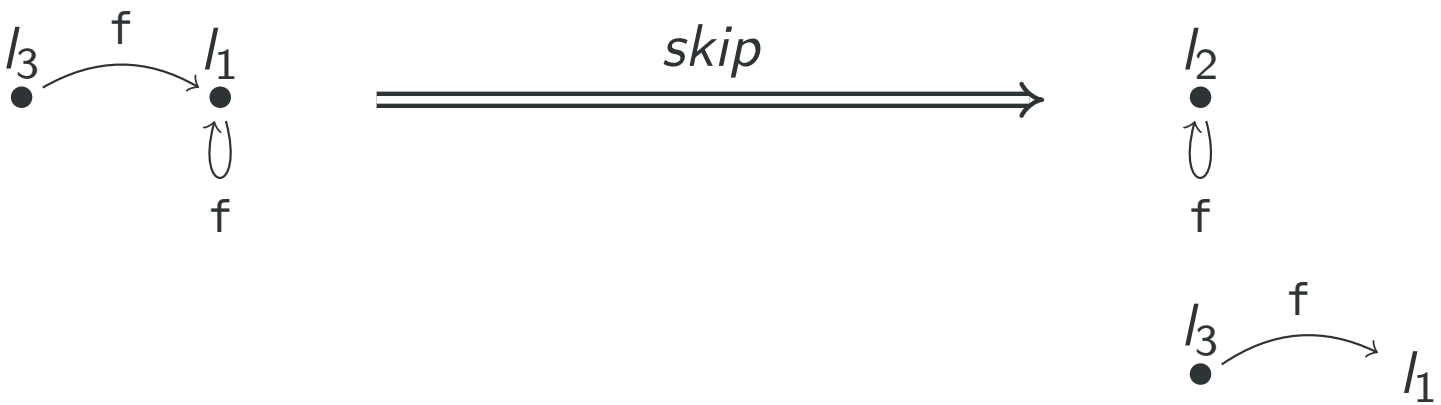
The Frame Rule

$$\textsc{glue-frame}\ \frac{\phi, p \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, p \Downarrow^{\sharp} \phi' \star \phi_c}$$

$$\dfrac{\dfrac{}{l_1 \mapsto \{\mathsf{f} : l_1\}, \mathit{skip} \Downarrow^{\sharp} l_1 \mapsto \{\mathsf{f} : l_1\}}\ \textsc{red-skip}}{l_1 \mapsto \{\mathsf{f} : l_1\}, \mathit{skip} \Downarrow^{\sharp} l_2 \mapsto \{\mathsf{f} : l_2\}}\ \textsc{glue-weaken}$$

The Frame Rule

$$\textsc{glue-frame}\ \frac{\phi, p \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, p \Downarrow^{\sharp} \phi' \star \phi_c}$$



$$\dfrac{\dfrac{\dfrac{}{l_1 \mapsto \{\mathsf{f} : l_1\}, skip \Downarrow^{\sharp} l_1 \mapsto \{\mathsf{f} : l_1\}}\ \textsc{red-skip}}{l_1 \mapsto \{\mathsf{f} : l_1\}, skip \Downarrow^{\sharp} l_2 \mapsto \{\mathsf{f} : l_2\}}\ \textsc{glue-weaken}}{l_1 \mapsto \{\mathsf{f} : l_1\} \star l_3 \mapsto \{\mathsf{f} : l_1\}, skip \Downarrow^{\sharp} l_2 \mapsto \{\mathsf{f} : l_2\} \star l_3 \mapsto \{\mathsf{f} : l_1\}}\ \textsc{glue-frame}$$

Membranes

My solution

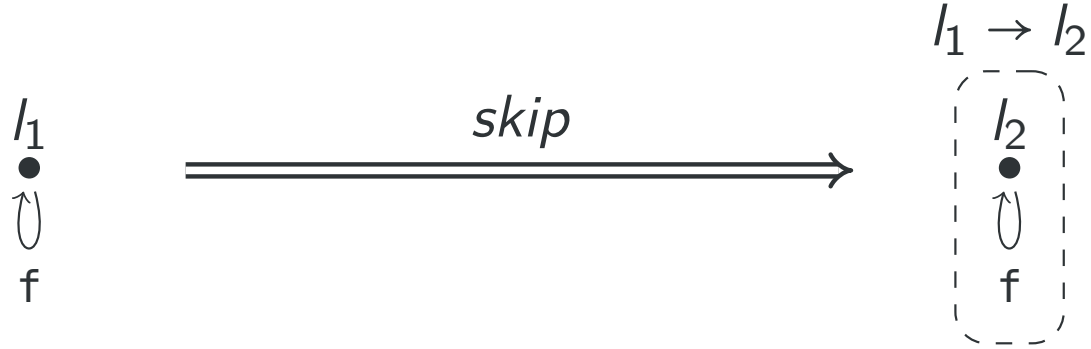
- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.

$$(\epsilon \mid l_1 \mapsto \{\mathtt{f} : l_1\}), \mathit{skip} \Downarrow^\sharp (\epsilon \mid l_1 \mapsto \{\mathtt{f} : l_1\})$$

Membranes

My solution

- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.

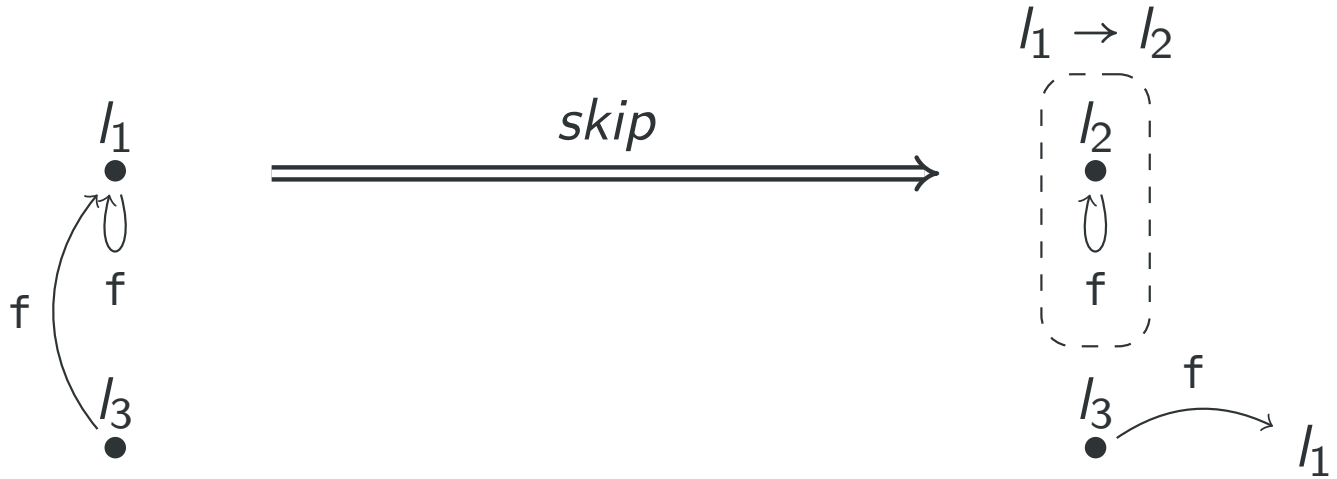


$$(\epsilon \mid l_1 \mapsto \{\mathtt{f} : l_1\}), \mathit{skip} \Downarrow^{\sharp} (l_1 \to l_2 \mid l_2 \mapsto \{\mathtt{f} : l_2\})$$

Membranes

My solution

- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.

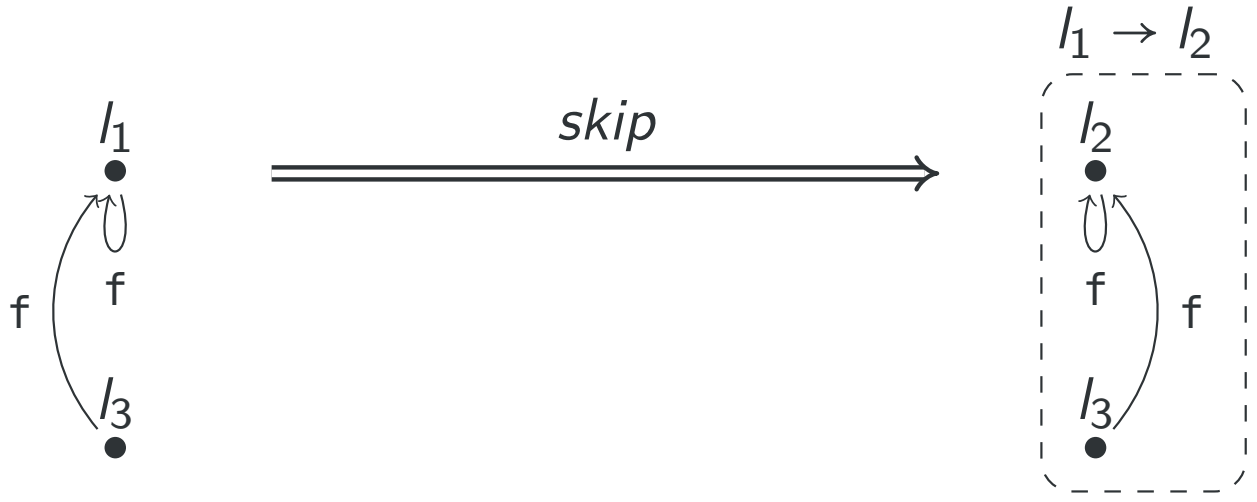


$$(\epsilon \mid l_1 \mapsto \{\mathtt{f} : l_1\}) \star l_3 \mapsto \{\mathtt{f} : l_1\}, \mathit{skip} \Downarrow^\sharp (l_1 \to l_2 \mid l_2 \mapsto \{\mathtt{f} : l_2\}) \star l_3 \mapsto \{\mathtt{f} : l_1\}$$

Membranes

My solution

- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.



$$(\epsilon \mid l_1 \mapsto \{\mathtt{f} : l_1\}) \star l_3 \mapsto \{\mathtt{f} : l_1\}, \mathit{skip} \Downarrow^{\sharp} (l_1 \to l_2 \mid l_2 \mapsto \{\mathtt{f} : l_2\} \star l_3 \mapsto \{\mathtt{f} : l_2\})$$

Membranes Solve Our Issue

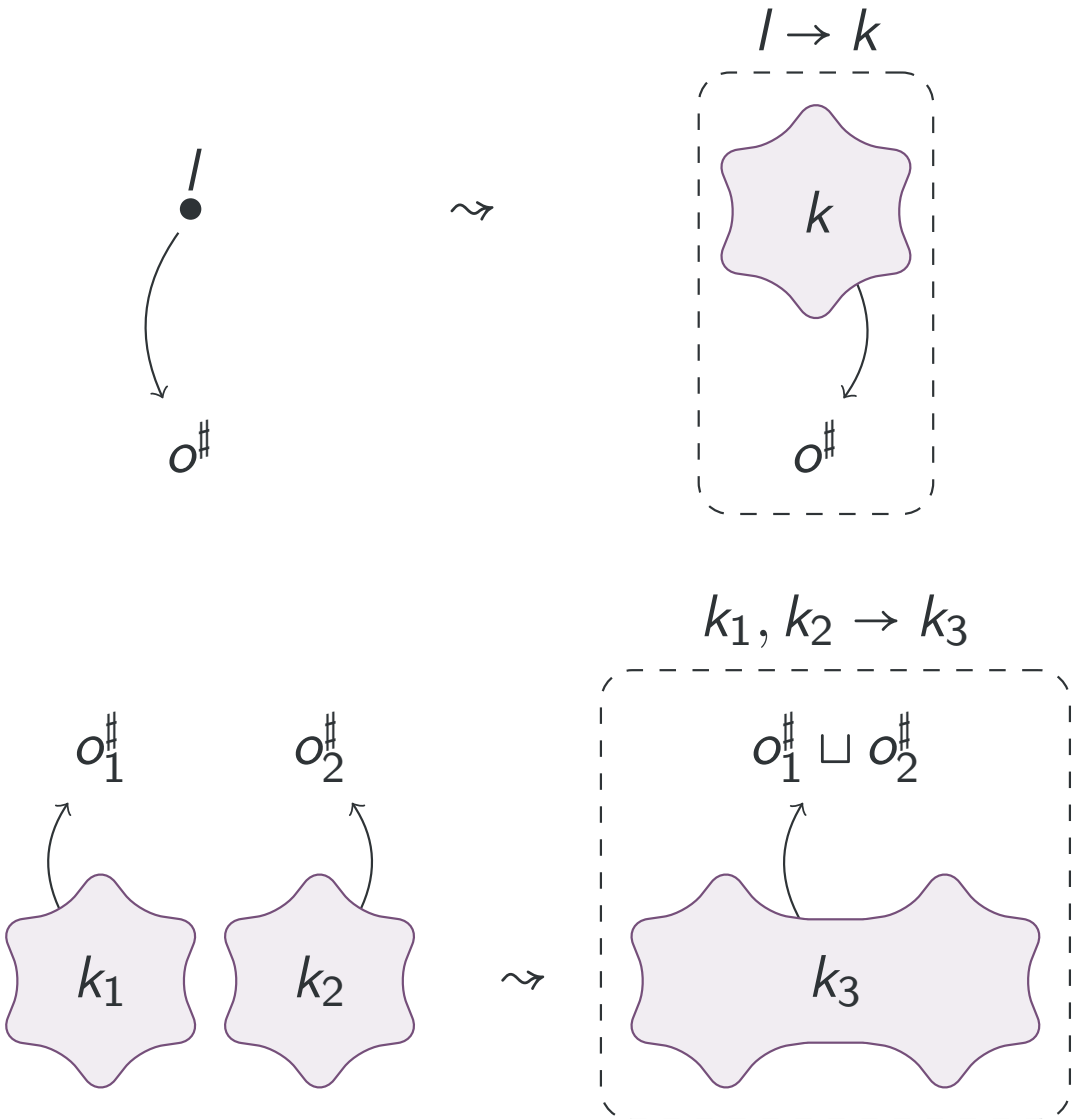
Theorem

The frame rule is sound in our framework, with membranes.

$$\textsc{glue-frame}\quad \frac{(M \mid \phi), p \Downarrow^{\sharp} (M' \mid \phi')}{(M \mid \phi \star M(\phi_c)), p \Downarrow^{\sharp} (M' \mid \phi' \star M'(\phi_c))}$$

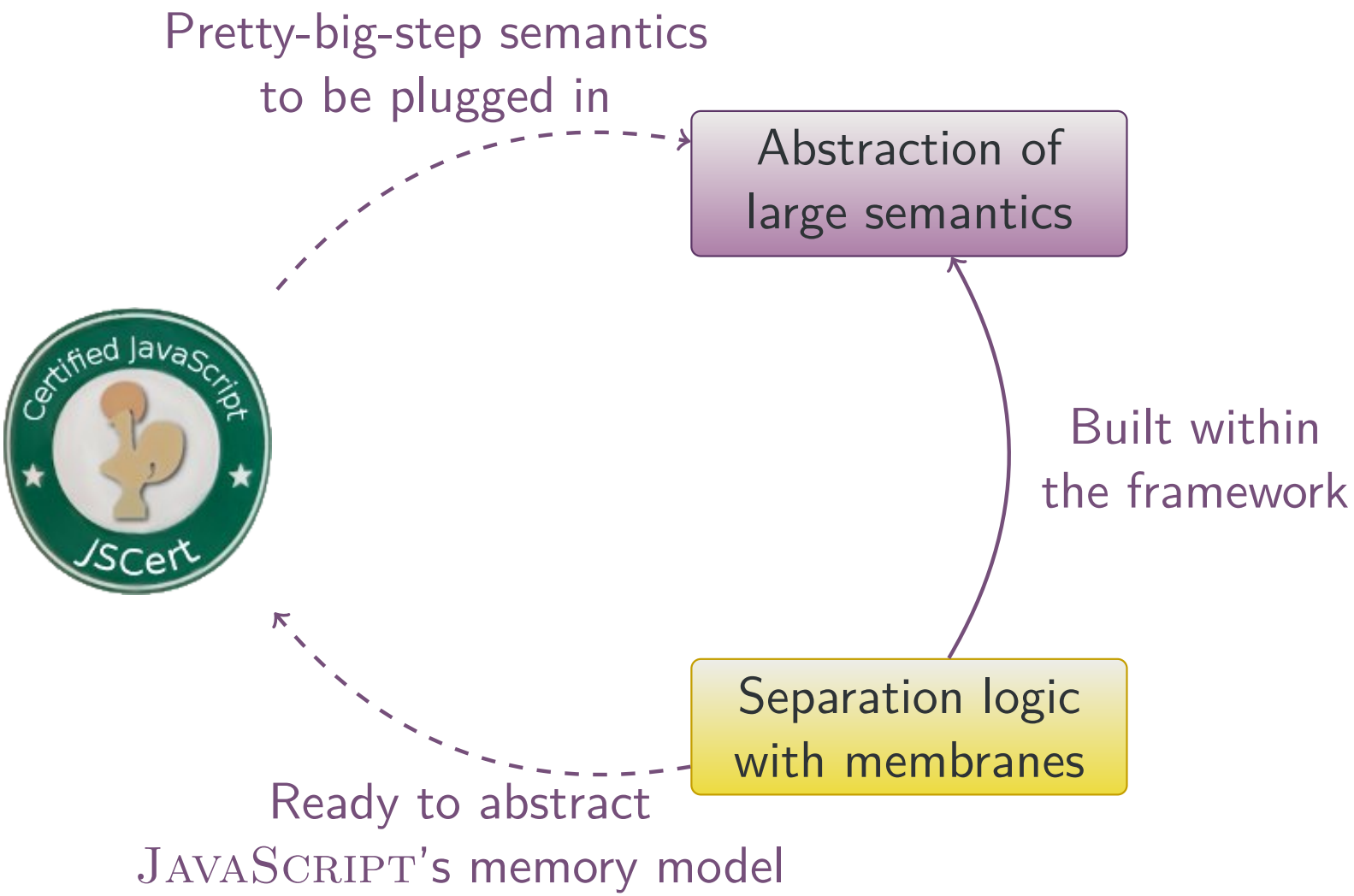
Membranes Can Introduce Approximations

Adding Summary Nodes, From Shape Analysis



Conclusion

General Situation



The JSCert Project

- Bridge between ECMAScript and test suites.
  - Bugs found in both sides.

- JSExplain and its interaction with the ECMAScript community.

Arthur Chargéraud, Alan Schmitt, and Thomas Wood. *Interactive Debugger for the* JavaScript *Specification*. 2016. URL: https://github.com/jscert/jsexplain.

- Already a basis for other projects.

Abstraction of Large Semantics

- A generic approach for building abstract semantics.
  - Not restricted to JavaScript!
- Genericity validated by the frame glue.

Future work: building analysers

- I already provide a generic verifier.
- Implement widening, narrowing, ...
- Prove existing analysers.

Thank You for Listening!


Martin Bodin et al. “A Trusted Mechanised JavaScript Specification”. In: *POPL*. 2014.


Martin Bodin, Thomas Jensen, and Alan Schmitt. “Pretty-big-step-semantics-based Certified Abstract Interpretation”. In: *JFLA*. 2014.


Martin Bodin, Thomas Jensen, and Alan Schmitt. “Certified Abstract Interpretation with Pretty-Big-Step Semantics”. In: *CPP*. 2015.


Martin Bodin, Thomas Jensen, and Alan Schmitt. “An Abstract Separation Logic for Interlinked Extensible Records”. In: *JFLA*. 2016.


Martin Bodin. *Thesis Companion*. 2016. URL: http://people.irisa.fr/Martin.Bodin/doktorigxo/companion.html?lang=en.

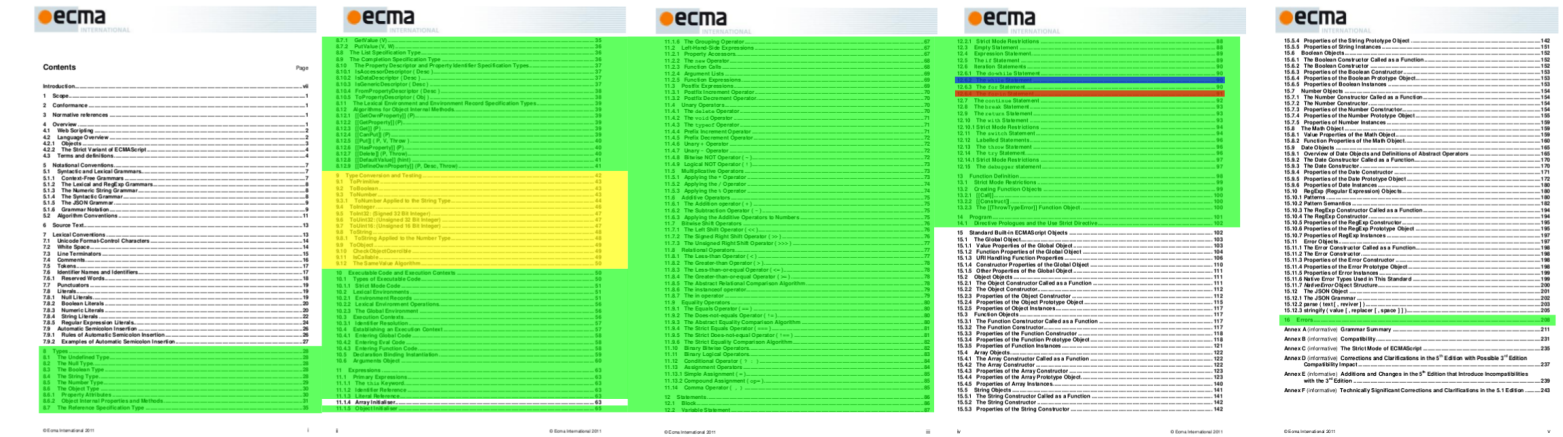
1. A Trustable Formal Semantics For JavaScript
2. Approximations of Language Semantics
3. Application to JavaScript

1. Why Formal Methods?
2. Trusting Programs
3. JavaScript
4. This Thesis
5. JSCert
6. Abstract Semantics
7. JavaScript Memory Model
8. Separation Logic
9. Abstract Domains
10. Conclusion

Bonuses

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

JSCert Coverage



1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Sequence in JSCert

```
1  Inductive red_stat : state → scope → stat → out → Prop :=
2
3  | red_stat_seq_1 : forall S C s1 s2 o1 o,
4    red_stat S C s1 o1 →
5    red_stat S C (seq_1 s2 o1) o →
6    red_stat S C (seq s1 s2) o
7
8  | red_stat_seq_2 : forall S C s2 o1,
9    abort o1 →
10   red_stat S C (seq_1 s2 o1) o1
11
12 | red_stat_seq_3 : forall S0 S C s2 o2 o,
13   red_stat S C s2 o2 →
14   red_stat S C (seq_2 o2) o →
15   red_stat S0 C (seq_1 s2 (out_ter S)) o
16
17 (* ... *).
```

$$\frac{S, C, s_1 \Downarrow o_1 \qquad o_1, seq_1\ s_2 \Downarrow o}{S, C, seq\ s_1\ s_2 \Downarrow o}\ \text{SEQ-}1(s_1, s_2)$$

$$\frac{}{o_1, seq_1\ s_2 \Downarrow o_1}\ \text{SEQ-}2(s_2)\quad \textbf{abort}\ o_1$$

$$\frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, seq_2 \Downarrow o}{o_1, seq_1\ s_2 \Downarrow o}\ \text{SEQ-}3(s_2)\quad \neg\textbf{abort}\ o_1$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Sequence in JSRef

- JSRef has been defined to be easily proven correct with respect to JSCert.

```
Definition run_seq      S C (s1 s2 : stat) : result :=
  if_success (run_stat      S C s1) (fun S1 o1 =>
    if_success (run_stat      S1 C s2) (fun S2 o2 =>
      (* ... *))).
```

```
Lemma run_seq_correct : forall      S C s1 s2,

  run_seq      S C s1 s2 = o →
  red_stat S C (stat_seq s1 s2) o.
Proof.
  introv    HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

Sequence in JSRef

- JSRef has been defined to be easily proven correct with respect to JSCert.

```
Definition run_seq runs S C (s1 s2 : stat) : result :=
  if_success (run_stat runs S C s1) (fun S1 o1 =>
    if_success (run_stat runs S1 C s2) (fun S2 o2 =>
      (* ... *))).
```

```
Lemma run_seq_correct : forall runs S C s1 s2,
  runs_type_correct runs →
  run_seq runs S C s1 s2 = o →
  red_stat S C (stat_seq s1 s2) o.
Proof.
  introv IH HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

runs

```
Record runs_type : Type := runs_type_intro {
    runs_type_stat : state → execution_ctx → stat → result ;
    runs_type_stat_while :
      state → resvalue → label_set → expr → stat → result ;
    (* ... *) }.
```

```
Fixpoint runs max_step : runs_type :=
  match max_step with
  | O =>
    { runs_type_stat := fun S => result_bottom S ;
      runs_type_stat_while := fun S => result_bottom S ;
      (* ... *) }
  | S max_step' =>  (* max_step = 1 + max_step' *)
    { runs_type_stat := fun S => run_stat (runs max_step') S ;
      runs_type_stat_while := fun S =>
        run_stat_while (runs max_step') S ;
      (* ... *) }
  end.
```

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Correctness with runs

```
1 Theorem run_javascript_correct_num : forall num p o,
2 run_javascript (runs num) p = result_out o →
3   red_javascript p o.
```

Martin Bodin and Alan Schmitt. "A Certified JavaScript Interpreter". In: *JFLA*. 2013.

Martin Bodin et al. "A Trusted Mechanised JavaScript Specification". In: *POPL*. 2014.

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Weaken to Glue

$$\textsc{Weaken}\ \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

- The $\textsc{Weaken}$ rule is not sound in a coinductive definition.

$$\dfrac{\dfrac{\dfrac{\vdots}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp} \qquad \sigma^\sharp \sqsubseteq \sigma^\sharp \quad \vdots \quad r^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}\ \textsc{Weaken} \qquad \sigma^\sharp \sqsubseteq \sigma^\sharp \quad \vdots \quad r^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}\ \textsc{Weaken}$$

Weaken to Glue

$$\textsc{Weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, \sigma^\sharp \sqsubseteq \sigma'^\sharp \wedge r'^\sharp \sqsubseteq r^\sharp \\ \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

Weaken to Glue

$$\textsc{Weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{l} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{l} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, \sigma^\sharp \sqsubseteq \sigma'^\sharp \wedge r'^\sharp \sqsubseteq r^\sharp \\ \quad \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{l} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, glue_n\left(\sigma'^\sharp, r'^\sharp, \sigma^\sharp, r^\sharp\right) \\ \quad \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

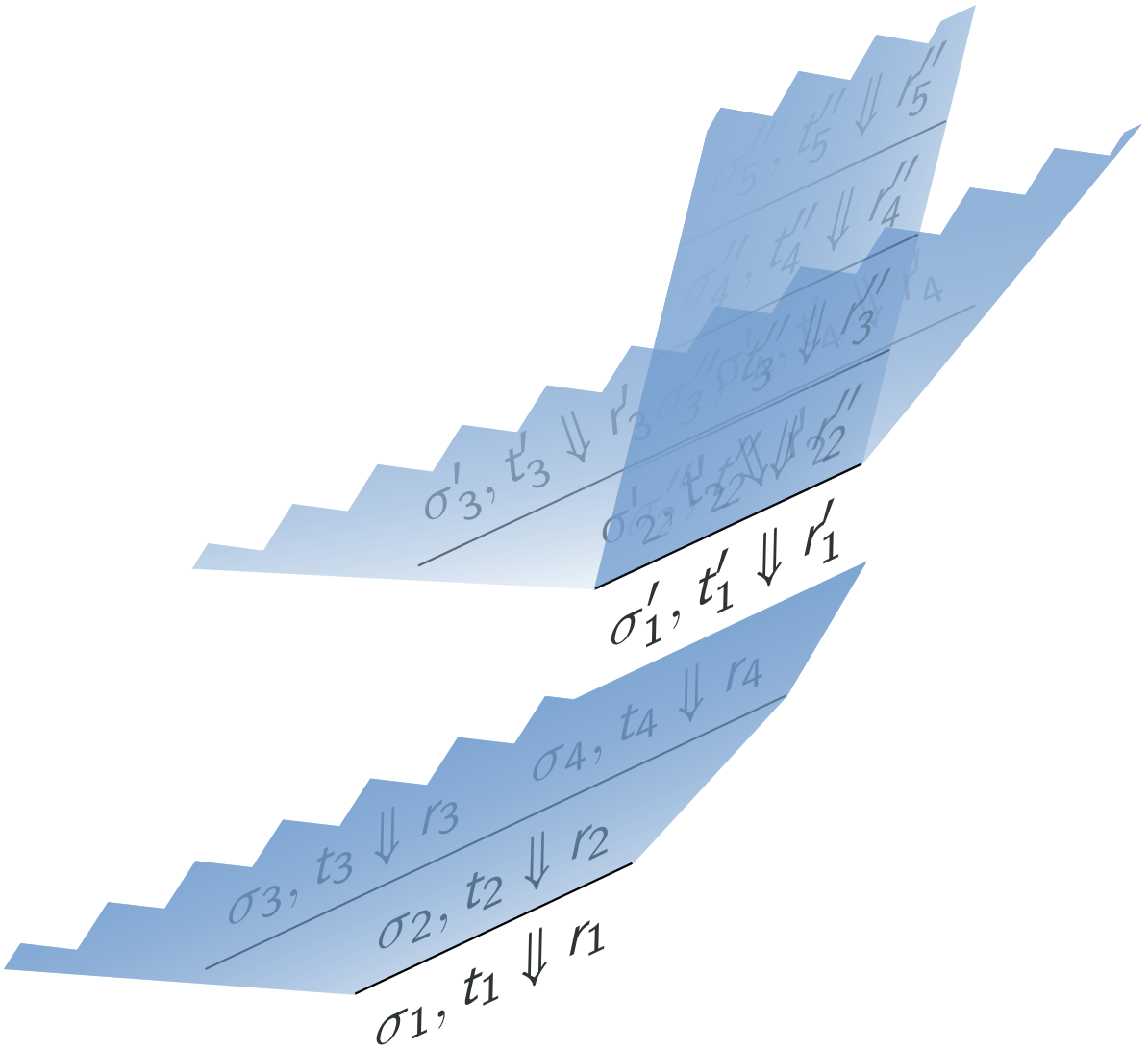
Immediate Consequence

$$F^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{l} \forall n.\, t = term(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, glue\left(\sigma'^\sharp, r'^\sharp, \sigma^\sharp, r^\sharp\right) \\ \quad \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$\begin{array}{l} apply_n^\sharp\left(\Downarrow_0\right) := \\ \left| \begin{array}{l} match\ rule^\sharp(n)\ \ with \\ \mid \quad Ax\left(ax^\sharp\right) \qquad\qquad \Rightarrow \left\{ \left(\sigma^\sharp, term(n), r^\sharp\right) \middle| ax^\sharp\left(\sigma^\sharp\right) = r^\sharp \right\} \\ \mid \quad R_1\left(up^\sharp\right) \qquad\qquad \Rightarrow \left\{ \left(\sigma^\sharp, term(i), r^\sharp\right) \middle| \begin{array}{r} up^\sharp\left(\sigma^\sharp\right) = \sigma'^\sharp \\ \wedge \sigma'^\sharp, term_1 \Downarrow_0 r^\sharp \end{array} \right\} \\ \mid \quad R_2\left(up^\sharp, next^\sharp\right) \Rightarrow \left\{ \left(\sigma^\sharp, term(n)\, r^\sharp\right) \middle| \begin{array}{r} up^\sharp\left(\sigma^\sharp\right) = \sigma'^\sharp \\ \wedge \sigma'^\sharp, term_1(n) \Downarrow_0 r_1^\sharp \\ \wedge next^\sharp\left(\sigma^\sharp, r_1^\sharp\right) = \sigma''^\sharp \\ \wedge \sigma''^\sharp, term_2(n) \Downarrow_0 r^\sharp \end{array} \right\} \end{array} \right. \end{array}$$

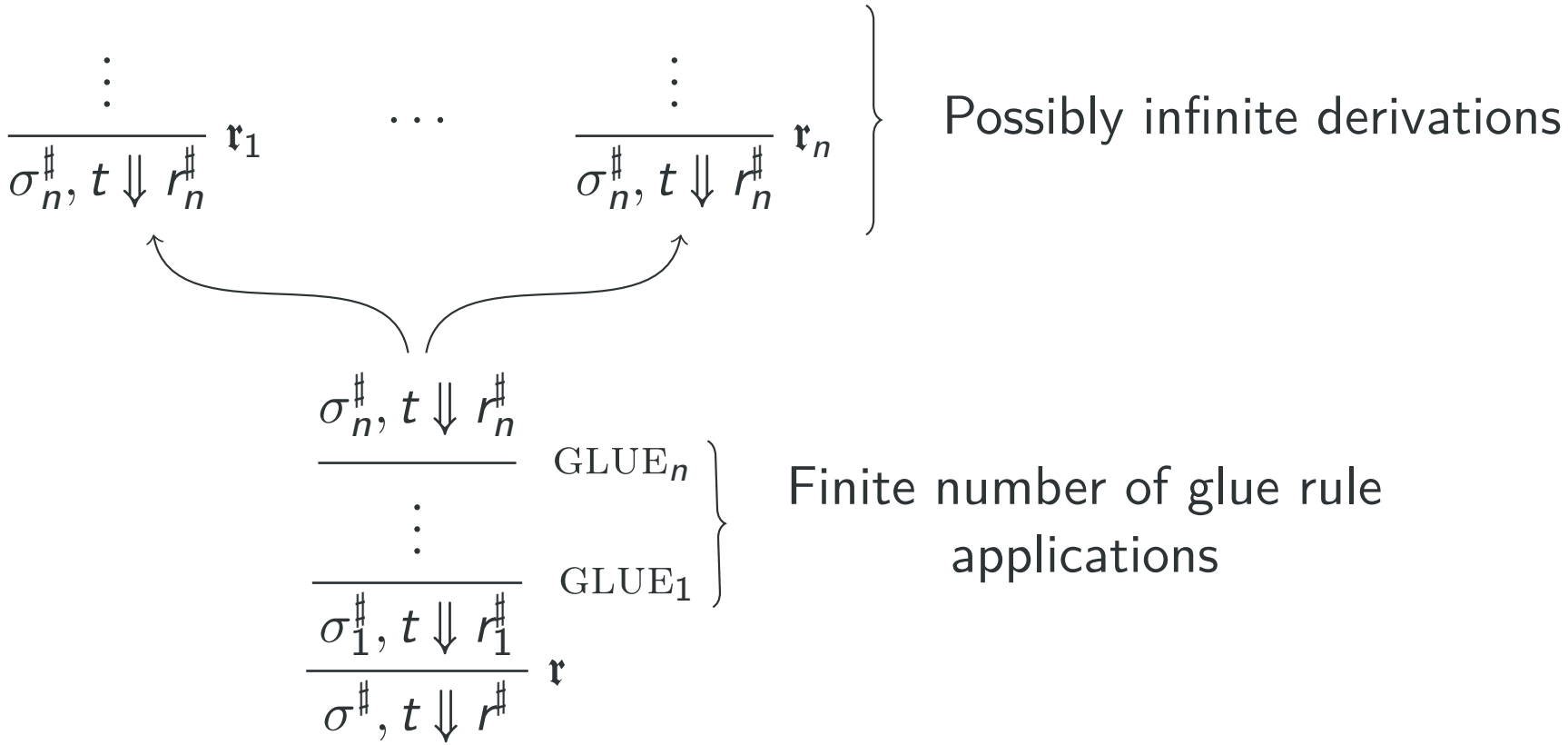
1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Illustration of the Abstract Derivations



1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Illustration of the Glue



1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Correctness of the Glue

k correctness

A semantic triple $\sigma^\sharp, t \Downarrow^\sharp r^\sharp$ is k correct, k being a number, if for any concrete derivation of depth less than k with conclusion $\sigma, t \Downarrow r$, then $\sigma \in \gamma\left(\sigma^\sharp\right)$ implies $r \in \gamma\left(r^\sharp\right)$.

Correctness of the glue

$$\forall \left(\sigma_i^\sharp\right), \left(r_i^\sharp\right). \mathit{glue}\left(\left\{\left(\sigma_i^\sharp, r_i^\sharp\right)\right\}, \sigma^\sharp, r^\sharp\right) \implies \left(\forall i.\ \sigma_i^\sharp, t \Downarrow^\sharp r_i^\sharp \text{ is } k \text{ correct}\right) \implies \sigma^\sharp, t \Downarrow^\sharp r^\sharp \text{ is } k \text{ correct}$$

Glue Correctness in Coq

```
1  Definition correct_up_to_depth k asigma ar :=
2    forall n sigma r (A : apply n sigma r),
3      gst asigma sigma →
4      cond n sigma →
5      apply_depth A < k →
6      gres ar r.
7
8  Definition glue_correct := forall P asigma ar k,
9    glue P asigma ar →
10   (forall asigma' ar',
11     P asigma' ar' →
12     correct_up_to_depth k asigma' ar') →
13   correct_up_to_depth k asigma ar.
```

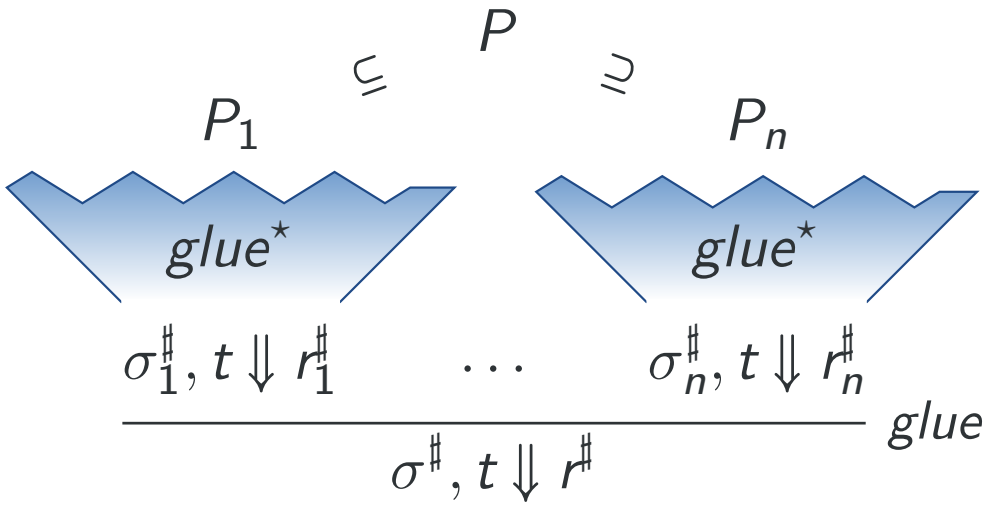
1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Examples of Glue Rules

$$\textsc{glue-weaken}\ \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, p \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, p \Downarrow^\sharp r^\sharp}$$

$$\textsc{glue-trace-partitioning}\ \frac{\sigma_1^\sharp, p \Downarrow^\sharp r^\sharp \qquad \ldots \qquad \sigma_n^\sharp, p \Downarrow^\sharp r^\sharp}{\sigma^\sharp, p \Downarrow^\sharp r^\sharp} \quad \gamma\left(\sigma^\sharp\right) \subseteq \gamma\left(\sigma_1^\sharp\right) \cup \ldots \cup \gamma\left(\sigma_n^\sharp\right)$$

Iterating glue




```
Inductive glue_iter :
    name → (ast → ares → Prop) → (ast → ares → Prop) :=
  | glue_iter_refl : forall n P asigma ar,
    P asigma ar → glue_iter n P asigma ar
  | glue_iter_cons : forall n P P' asigma3 ar3,
    (forall asigma2 ar2,
      P' asigma2 ar2 →
      glue_iter n P asigma2 ar2) →
    glue n P' asigma3 ar3 →
    glue_iter n P asigma3 ar3.
```

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

6×7

$$\mathtt{a} := 6; \mathtt{b} := 7; \mathtt{r} := 0; \mathtt{n} := a; \mathit{while}\, n\, (\mathtt{r} := r + b; \mathtt{n} := n - 1)$$

$$(\{r \mapsto +, b \mapsto +, a \mapsto +, n \mapsto \top\}, \bot)$$

6×7

$$\mathtt{a} := 6; \mathtt{b} := 7; \mathit{prod}\,(n) \;:=\; \{\mathit{if}\, n\; (\mathit{prod}\,(n-1)\,; \mathtt{r} := r + b)\,(\mathtt{r} := 0)\,\}; \mathit{prod}\,(a)$$

$$(\{r \mapsto +, b \mapsto +, a \mapsto +\}, \bot)$$

6×7

$$\mathtt{a} := 6; \mathtt{b} := 7; \mathit{prod}\,(n) \;:= \{\mathit{if}\, n\; (\mathit{prod}\,(n-1)\,; \mathtt{r} := \mathbf{r} + \mathbf{b})\,(\mathtt{r} := \mathbf{0})\,\}; \mathit{prod}\,(a)$$

$$(\{r \mapsto +, b \mapsto +, a \mapsto +\}, \bot)$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Abstract Objects

$$o^\sharp : \mathit{Field} \rightharpoonup \mathit{Val}^\sharp$$

Track which fields are present

A special abstract value $\boxtimes$ denoting the absence of fields.

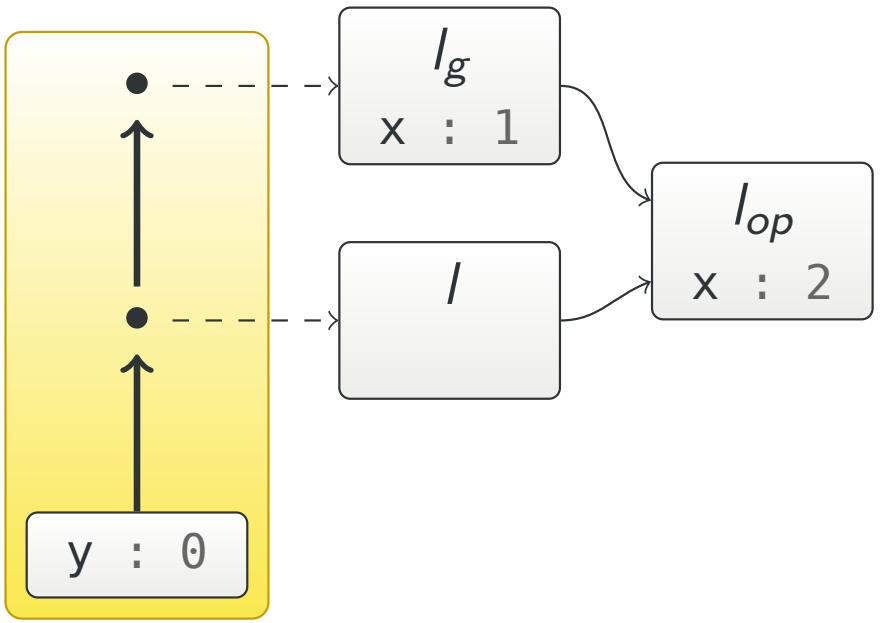
Mixing different types of values in the same fields

$$\left.\begin{array}{r} +, \pm, -, \ldots \in v^\sharp \\ l \in v^\sharp \\ \boxtimes \in v^\sharp \end{array}\right\} \text{Basic values can be mixed: } \boxtimes \sqcup l_1 \sqcup l_2 \sqcup +.$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^{\sharp}$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

JavaScript Memory Model: Reading

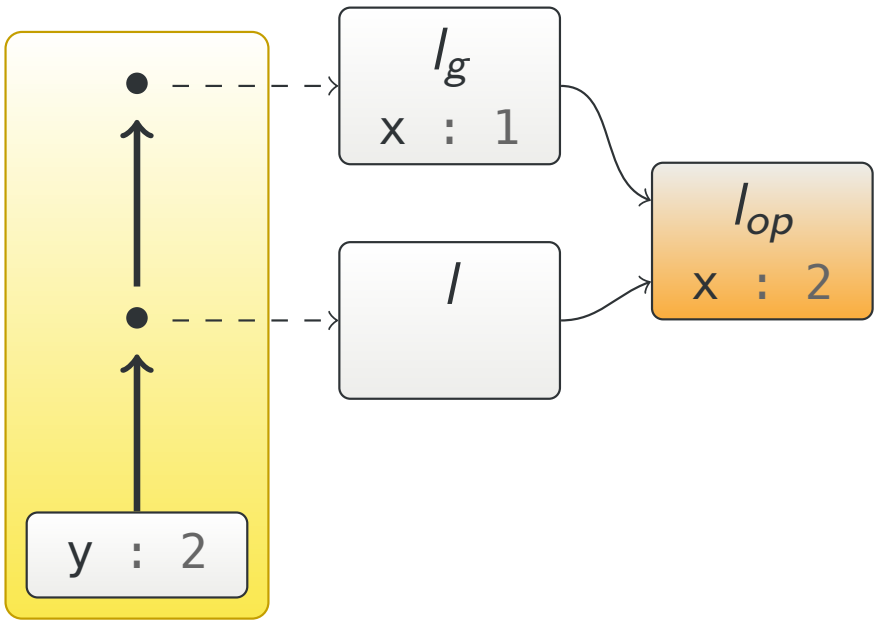
```
1  x = 1 ;
2  Object.prototype.x = 2 ;
3  with (new Object ()){
4    function (y){
5      /* ... */
6      y = x
```



Scope chain

JavaScript Memory Model: Reading

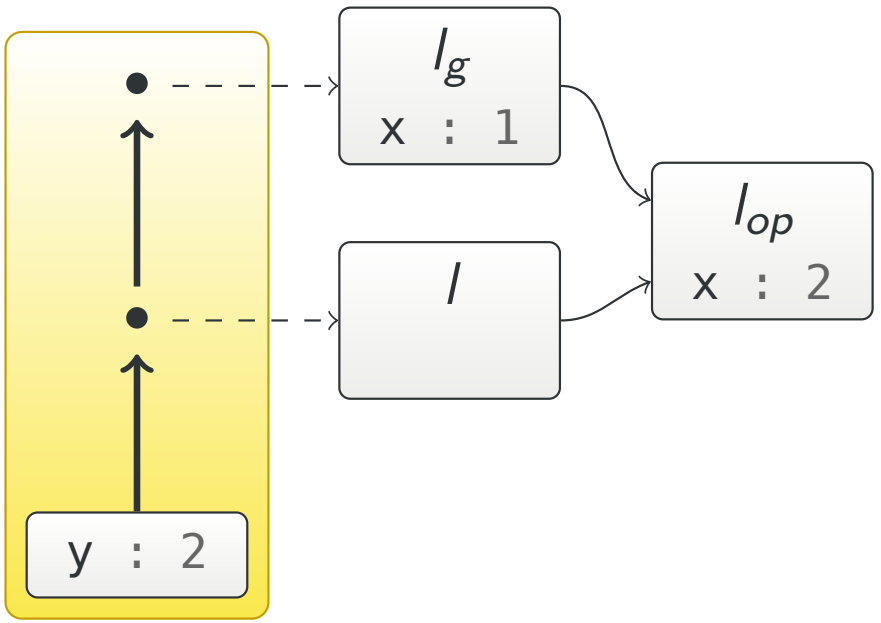
```
1  x = 1 ;
2  Object.prototype.x = 2 ;
3  with (new Object ()){
4    function (y){
5      /* ... */
6      y = x
```



Scope chain

JavaScript Memory Model: Writing

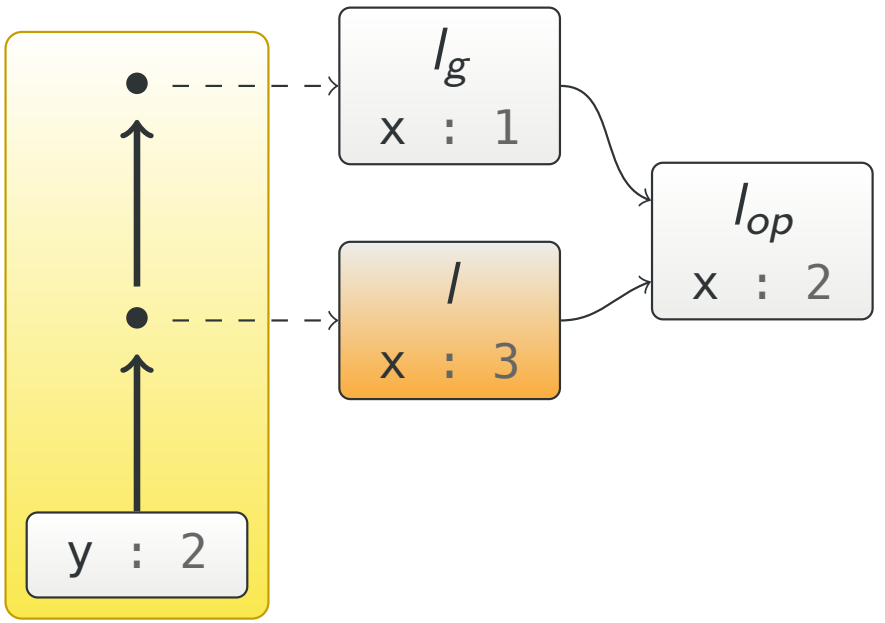
```
1  x = 1 ;
2  Object.prototype.x = 2 ;
3  with (new Object ()){
4    function (y){
5      /* ... */
6      x = 3
```



Scope chain

JavaScript Memory Model: Writing

```
1  x = 1 ;
2  Object.prototype.x = 2 ;
3  with (new Object ()){
4    function (y){
5      /* ... */
6      x = 3
```



Scope chain

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

An Issue About 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.

An Issue About JavaScript

An Introduction to Separation Logic, by John C. Reynolds

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

- JavaScript's `delete` does exactly this.

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Materialisation

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

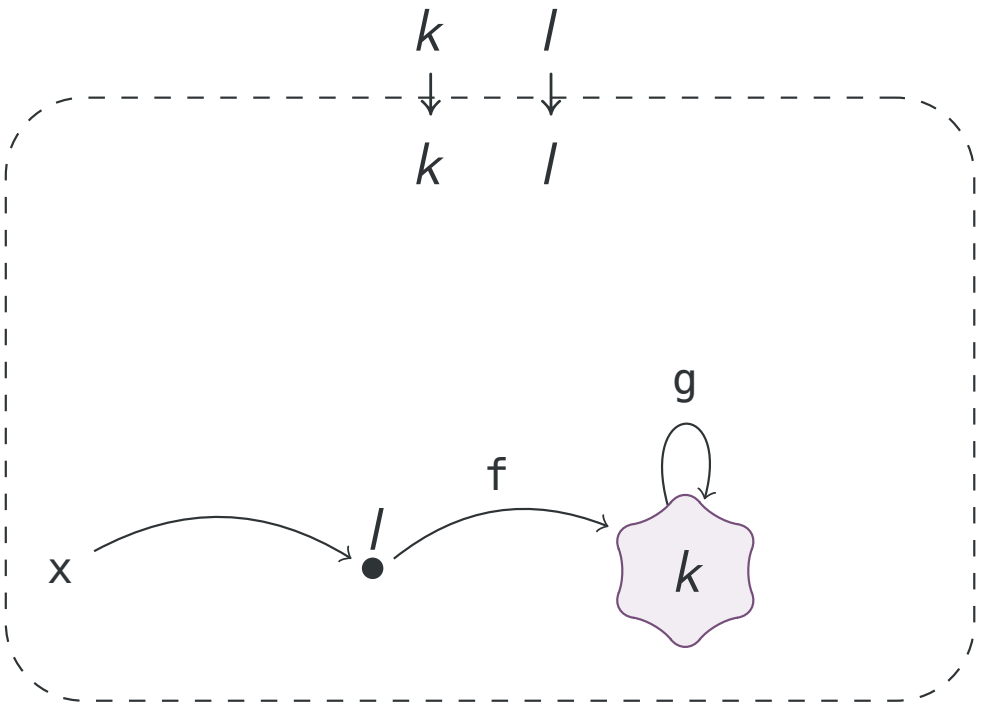
A Weak Update From a Strong Specification

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\left(\epsilon \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}, l_i', v^\sharp\right), asn \Downarrow \left(\epsilon \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; v^\sharp\right\}\right)}\,\textsc{field-asn}(\mathtt{f})}{\left(\epsilon \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\} \star k_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}, l_i', v^\sharp\right), asn \Downarrow \left(\epsilon \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; v^\sharp\right\} \star k_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}\right)}\,\textsc{glue-frame}}{\left(k_o \to l_i' + k_i' \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\} \star k_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}, l_i', v^\sharp\right), asn \Downarrow \left(k_o \to l_i' + k_i' \;\middle|\; l_i' \mapsto \left\{\mathtt{f}:\; v^\sharp\right\} \star k_i' \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}\right)}\,\textsc{glue-frame}}{\left(k_o \to k_i \;\middle|\; k_i \mapsto \left\{\mathtt{f}:\; u^\sharp\right\}, k, v^\sharp\right), asn \Downarrow \left(k_o \to k_i \;\middle|\; k \mapsto \left\{\mathtt{f}:\; u^\sharp \sqcup v^\sharp\right\}\right)}\,\textsc{glue-weaken}$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

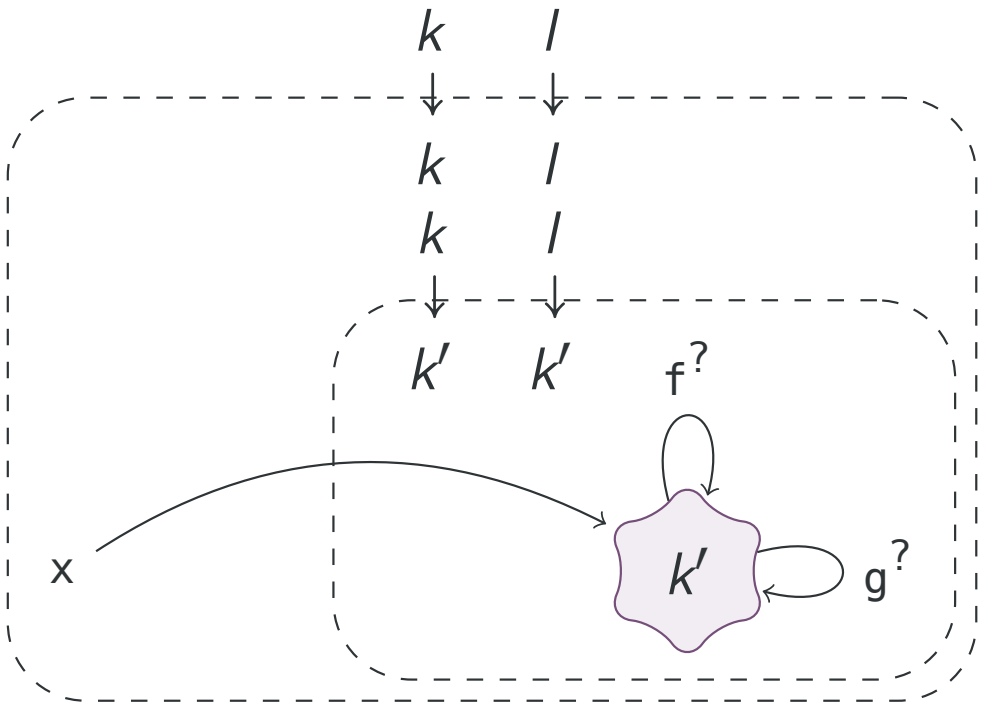
Membrane Manipulation

$$(k \to k, l \to l \mid \mathsf{x} \doteq l \star l \mapsto \{\mathsf{f} : k, \overline{\boxtimes} :\} \star k \mapsto \{\mathsf{g} : k, \overline{\boxtimes} :\})$$



Membrane Manipulation

$$(k \to k, l \to l \mid \mathsf{x} \doteq l \star l \mapsto \{\mathsf{f} : k, \overline{\boxtimes} :\} \star k \mapsto \{\mathsf{g} : k, \overline{\boxtimes} :\})$$

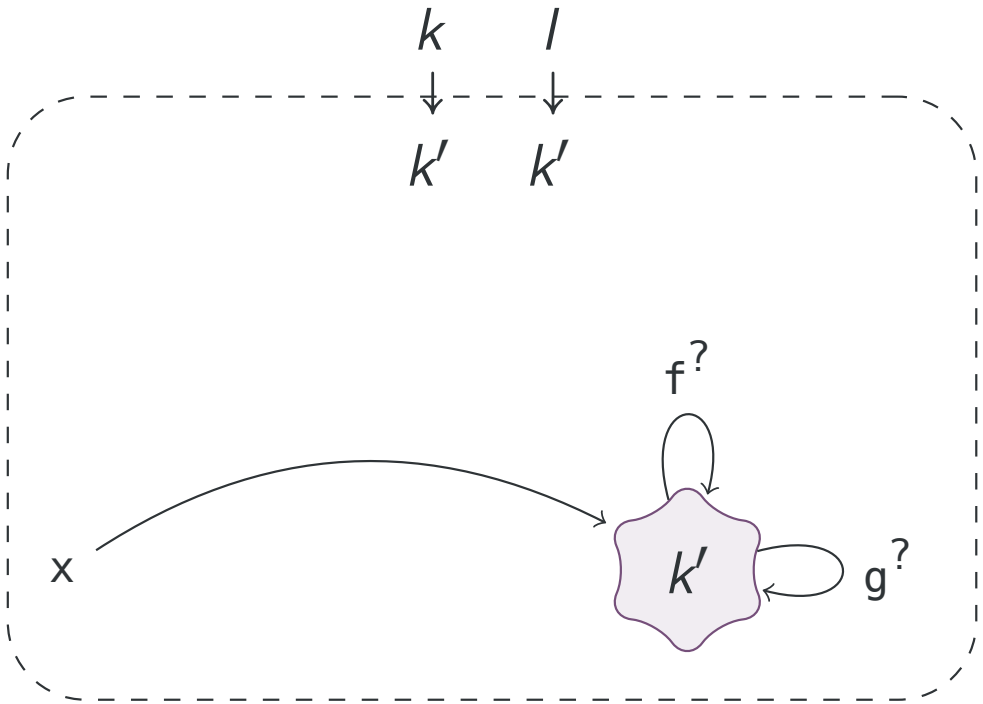


$$(k \to k, l \to l \mid \mathsf{x} \doteq l)$$

$$\circledast \left(k \to k', l \to k' \;\middle|\; k' \mapsto \{\mathsf{f} : k' \sqcup \boxtimes, \mathsf{g} : k' \sqcup \boxtimes, \overline{\boxtimes} :\}\right)$$

Membrane Manipulation

$$(k \to k, l \to l \mid \mathsf{x} \doteq l \star l \mapsto \{\mathsf{f} : k, \overline{\boxtimes} :\} \star k \mapsto \{\mathsf{g} : k, \overline{\boxtimes} :\})$$

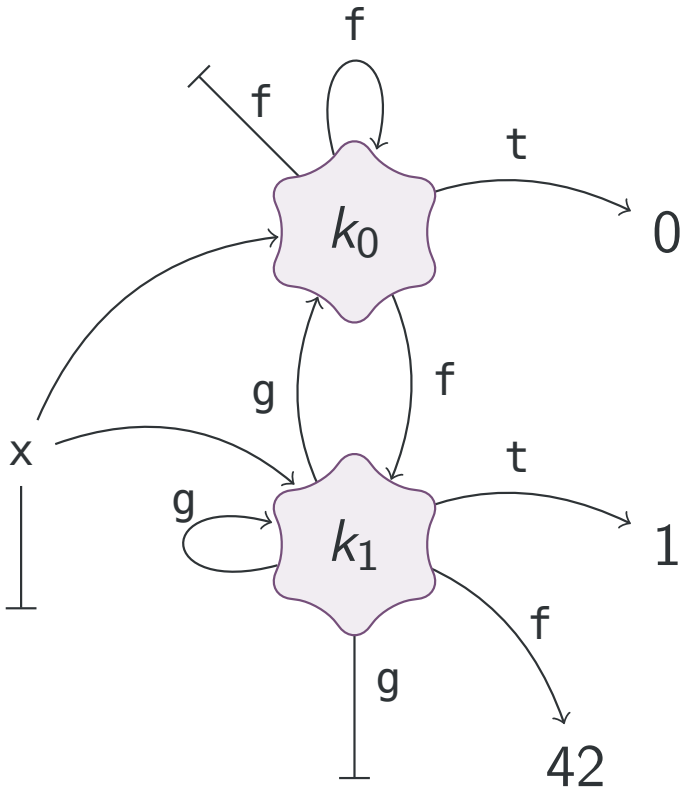


$$\left(k \to k', l \to k' \mid \mathsf{x} \doteq k' \star k' \mapsto \left\{\mathsf{f} : k' \sqcup \boxtimes, \mathsf{g} : k' \sqcup \boxtimes, \overline{\boxtimes} :\right\}\right)$$

1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

Loop Invariant

```
x := nil;
while random(
    t := x;
    x := alloc;
    if random (
        x.t := 0;
        x.f := t
    )(
        x.t := 1;
        x.g := t;
        x.f := 42
    )
);
while x ≠ nil(
    if x.t (x := x.g)(x := x.f)
)
```



1. JSCert's Coverage,
2. Sequence in JSCert,
3. Detailed sequence in JSRef,
4. runs,
5. Correctness with runs,
6. Correctness theorem for $\Downarrow^\sharp$,
7. Weaken to Glue,
8. Immediate Consequence,
9. Illustration of abstract derivations,
10. Illustration of the glue,
11. Correctness of the glue,
12. Some glue rules,
13. 6×7,
14. Abstract Objects,
15. Scope chain in JavaScript,
16. Reynolds,
17. Materialisation,
18. A weak update from a strong specification,
19. Bigger example of membrane manipulation,
20. Overcomplicated loop invariant,
21. Rule GLUE-FRAME.

1. A Trustable Formal Semantics For JavaScript
2. Approximations of Language Semantics
3. Application to JavaScript