

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”

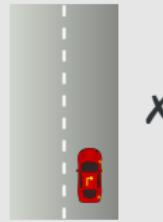
What Is a Bug?

- A bug is an unwanted behaviour.

Expected behaviour



Unwanted behaviour



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

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.

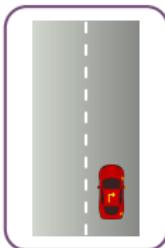
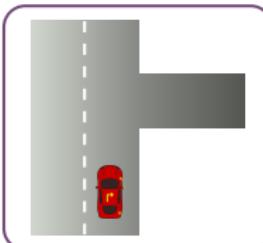
Bugs Yield Unwanted States

Program

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

Specification

The car does not go out of the road.



States

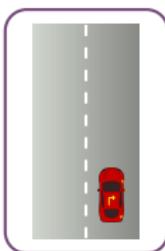
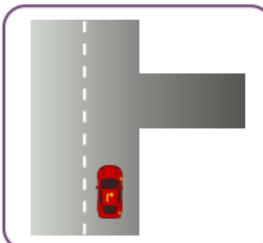
Bugs Yield Unwanted States

Program

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

Specification

The car does not go out of the road.



States

Unwanted States

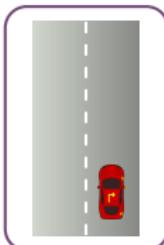
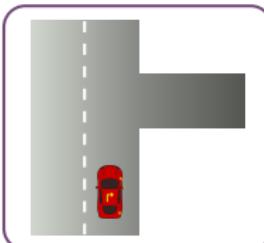
Bugs Yield Unwanted States

Program

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

Specification

The car does not go out of the road.

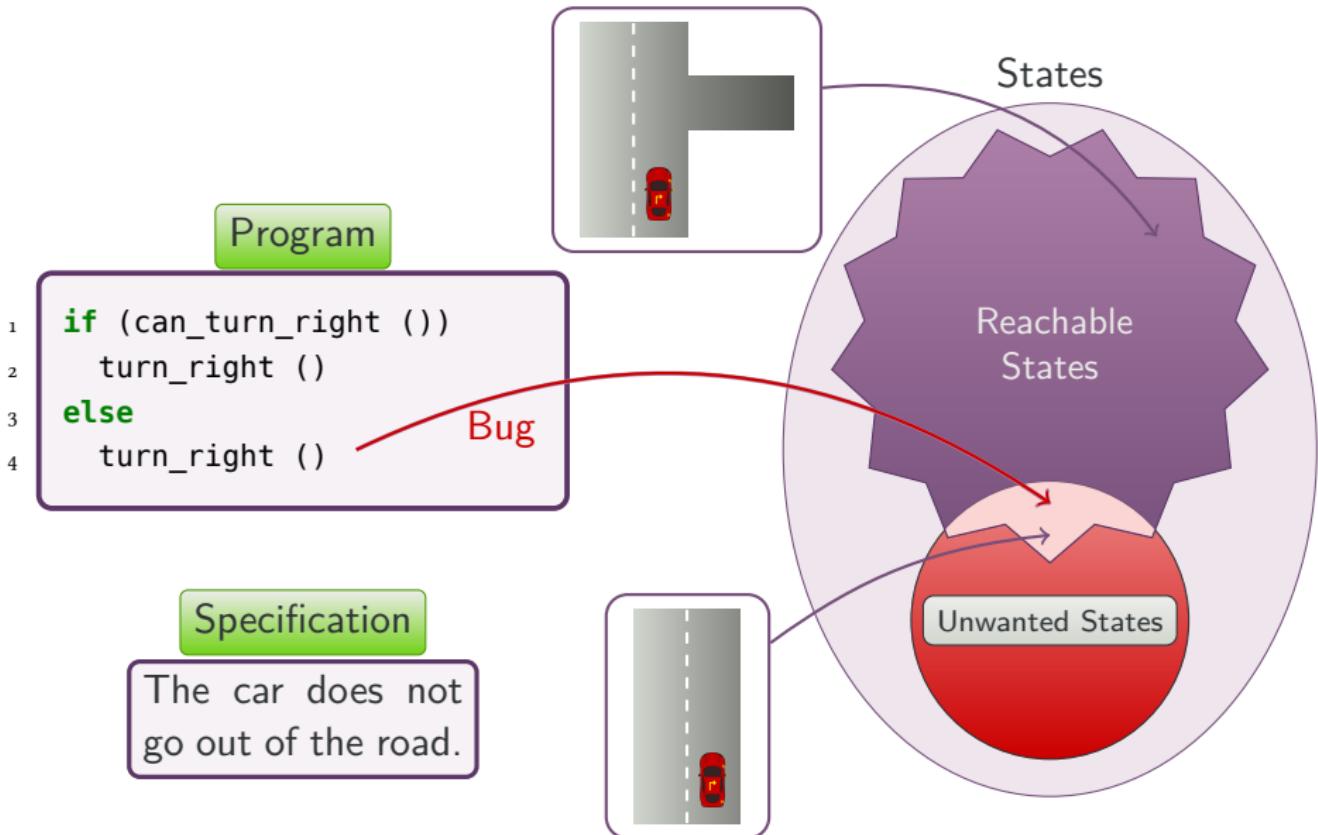


States

Reachable States

Unwanted States

Bugs Yield Unwanted States

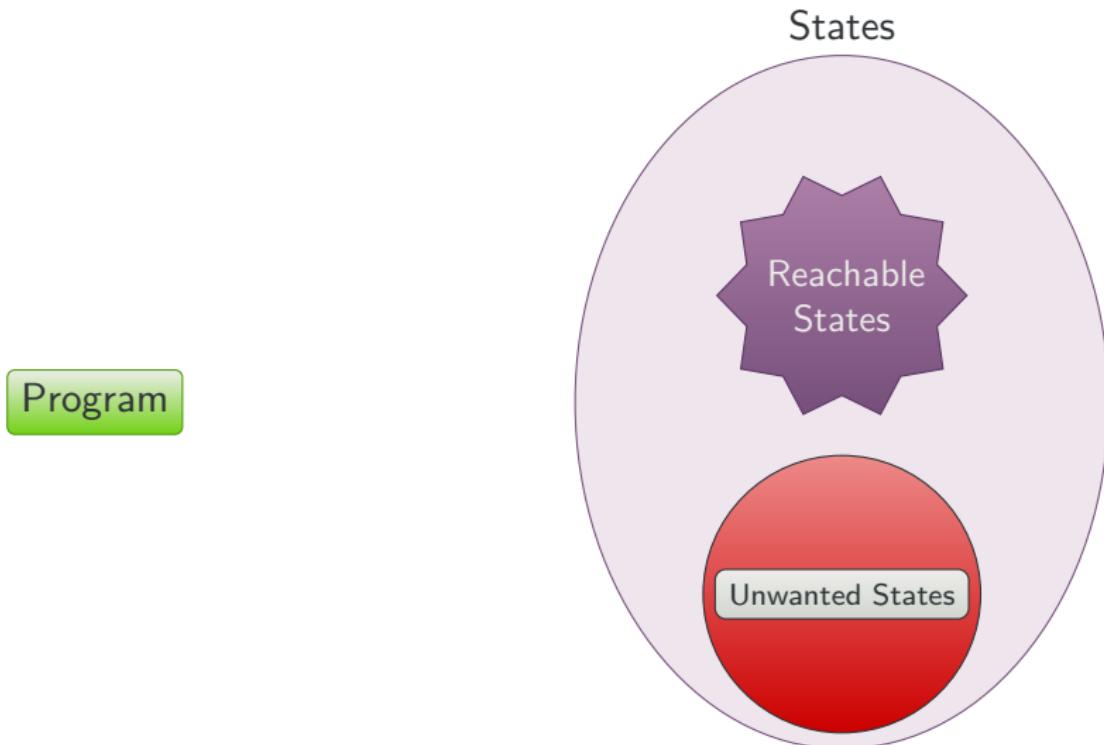


Are specifications correctly implemented?

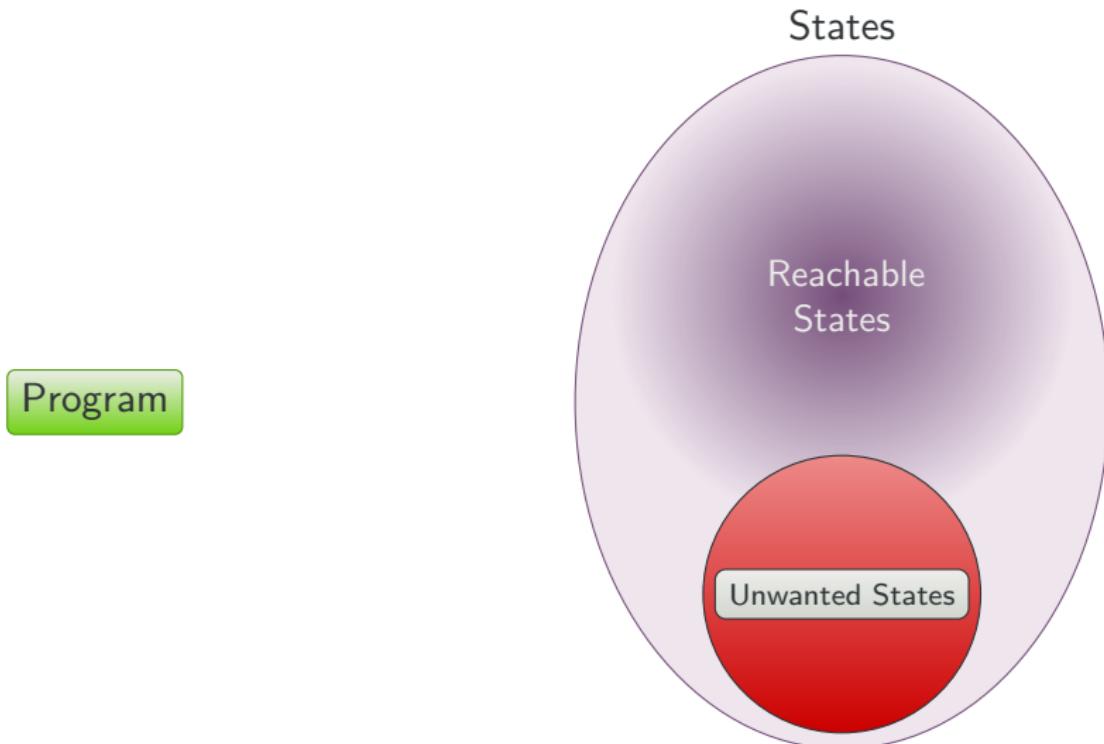
How can we avoid bugs?



Trusting Programs: Existing Methods

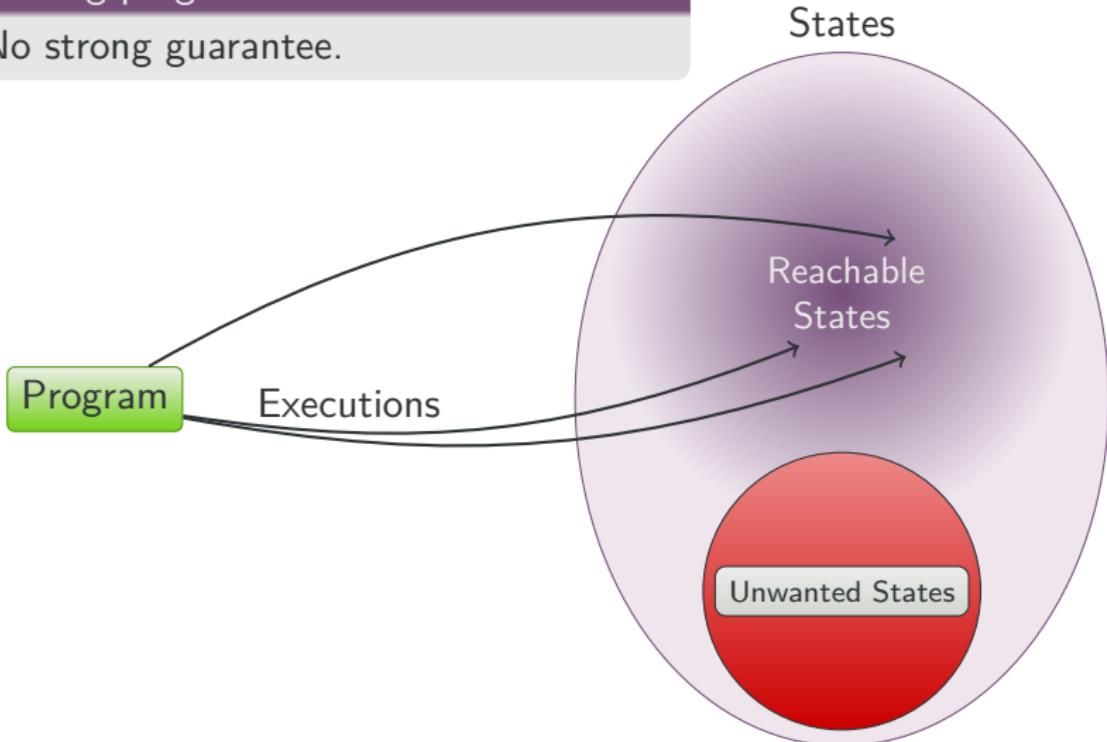


Trusting Programs: Existing Methods



Trusting Programs: Existing Methods

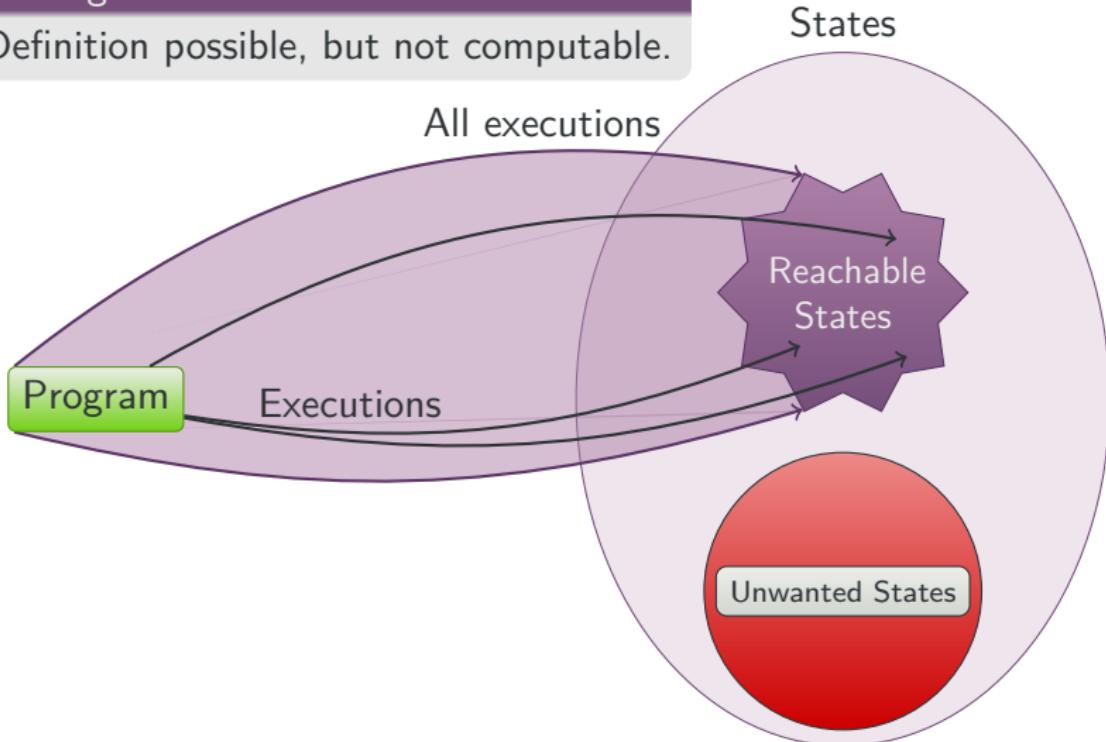
Testing programs
No strong guarantee.



Trusting Programs: Existing Methods

Testing *all* instances

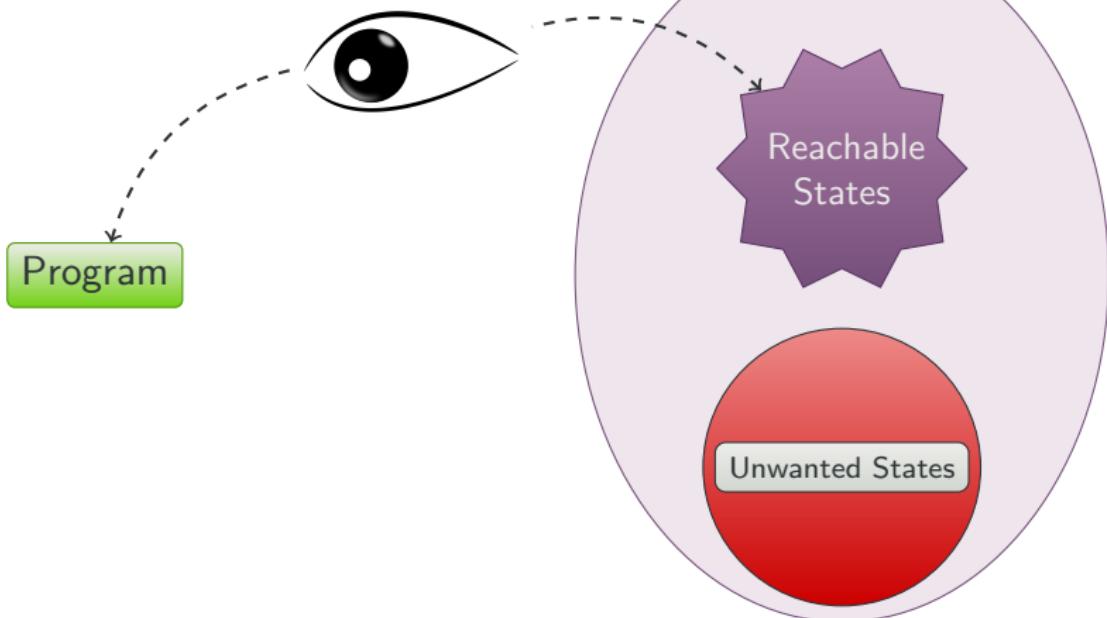
Definition possible, but not computable.



Trusting Programs: Existing Methods

Reviewing programs

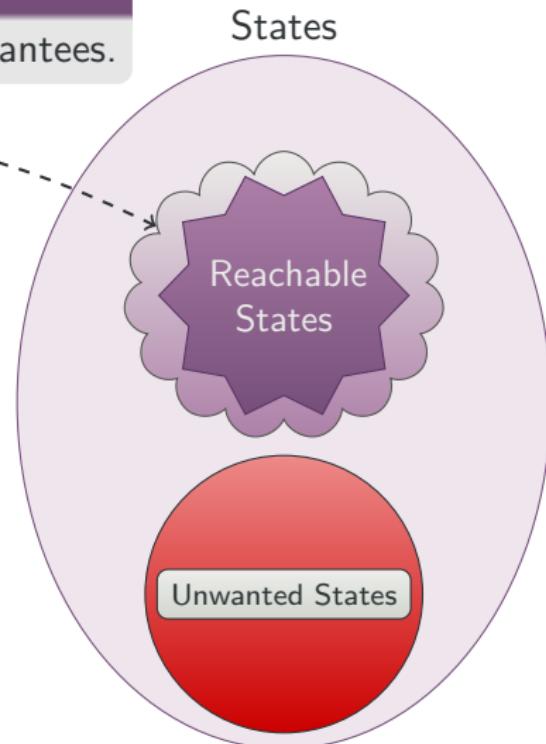
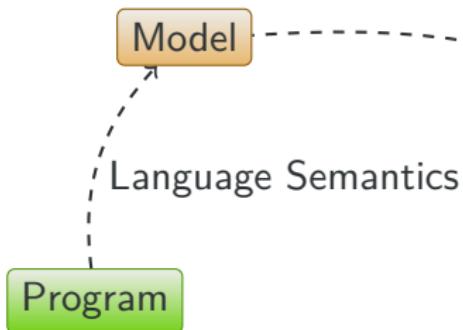
No strong guarantee.



Trusting Programs: Existing Methods

Proving programs

Complex, but provide strong guarantees.



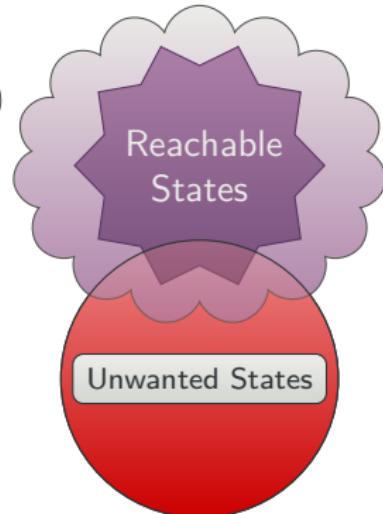
Approximations



Safety proven

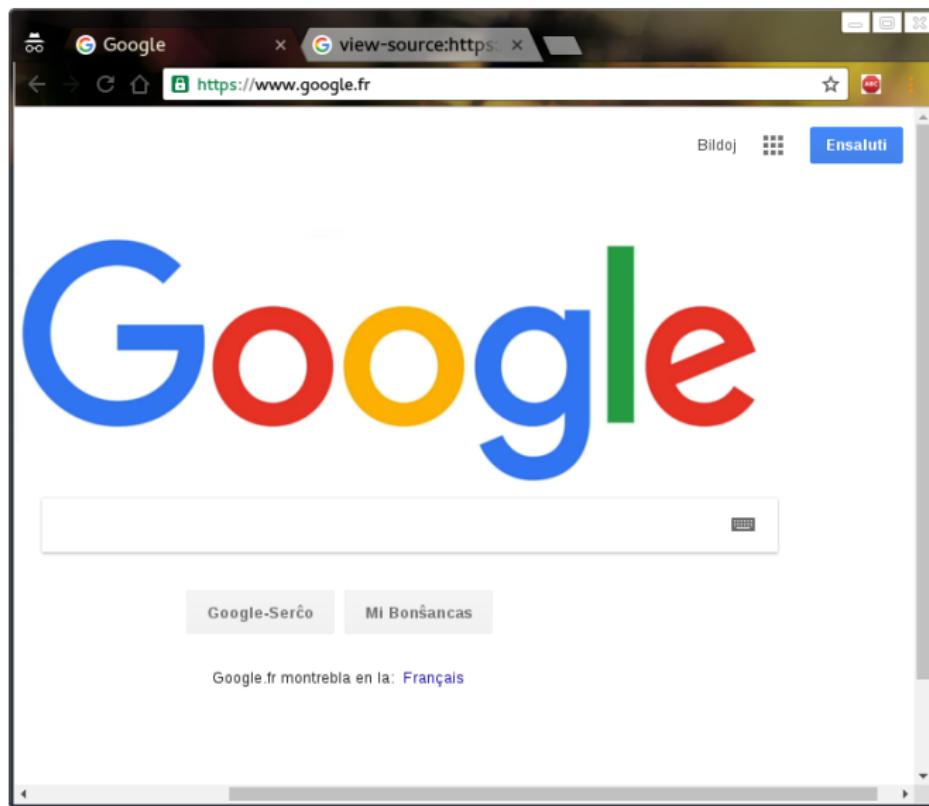


Unconclusive

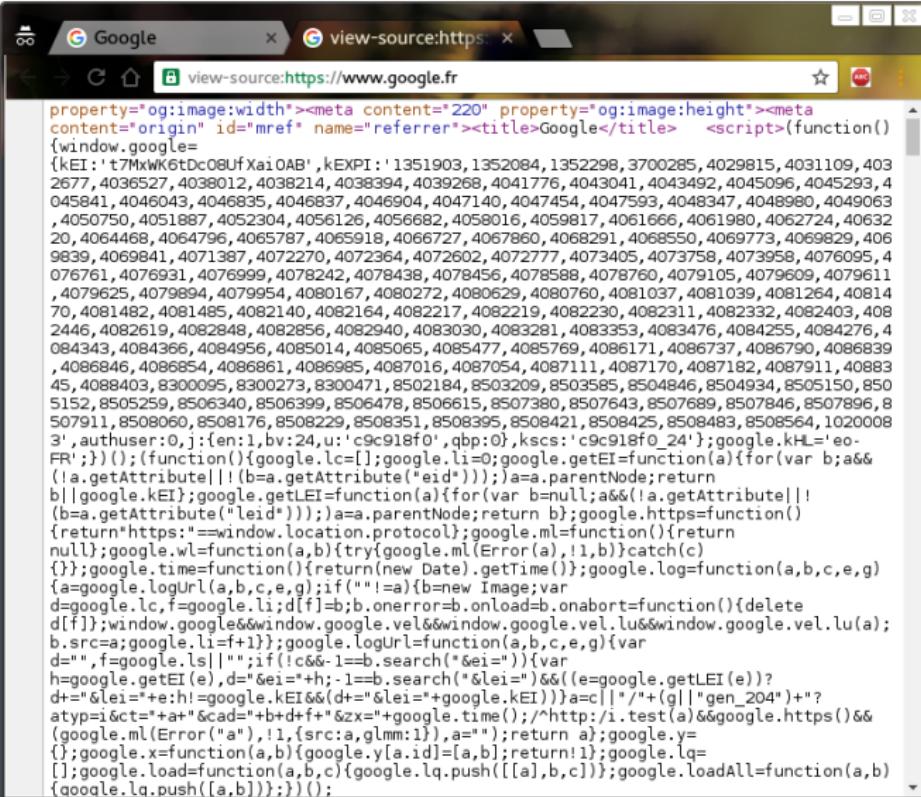


JAVASCRIPT

The Language of the Web



The Language of the Web

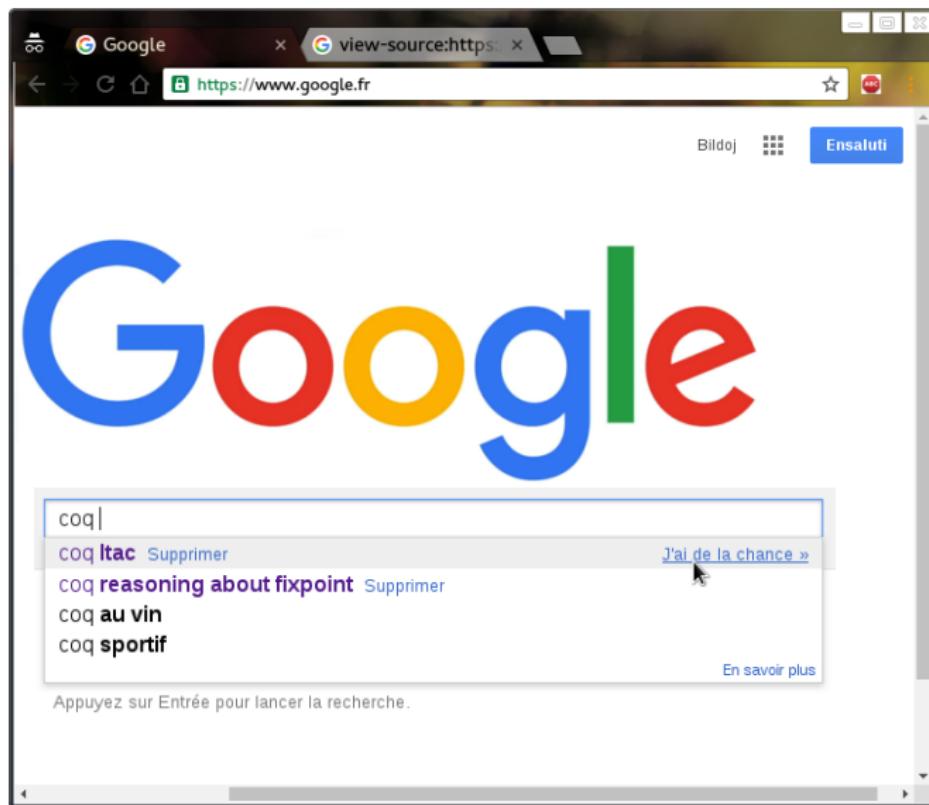


A screenshot of a web browser window. The address bar shows 'view-source:https://www.google.fr'. The main content area displays over 7,500 lines of JavaScript code, which is the source code for Google's homepage. The code is highly complex, containing numerous functions, loops, and conditional statements. It includes references to various Google services like Google Analytics and Google Ads.

```
property="og:image:width"><meta content="220" property="og:image:height"><meta content="origin" id="mref" name="referrer"><title>Google</title>    <script>(function() {window.google= {kEI:'t7MxwK6tDc08Ufxai0AB',kEXP:'1351903,1352084,1352298,3700285,4029815,4031109,4032677,4036527,4038012,4038214,4038394,4039268,4041776,4043041,4043492,4045096,4045293,4045841,4046043,4046835,4046837,4046904,4047140,4047454,4047593,4048347,4048980,4049063,4050750,4051887,4052304,4056126,4056682,4058016,4059817,4061666,4061980,4062724,4063220,4064468,4064796,4065787,4065918,4066727,4067860,4068291,4068550,4069773,4069829,4069839,4069841,4071387,4072270,4072364,4072602,4072777,4073405,4073758,4073958,4076095,4076671,4076931,4076999,4078242,4078438,4078456,4078588,4078760,4079105,4079609,4079611,4079625,4079894,4079954,4080167,4080272,4080629,4080760,4081037,4081039,4081264,4081470,4081485,4082140,4082164,4082217,4082219,4082230,4082311,4082332,4082403,4082446,4082619,4082848,4082856,4082940,4083030,4083281,4083353,4083476,4084255,4084276,4084343,4084366,4084956,4085014,4085065,4085477,4085769,4086171,4086737,4086790,4086839,4086846,4086854,4086861,4086985,4087016,4087054,4087111,4087170,4087182,4087911,4088345,4088403,8300095,8300273,8300471,8502184,8503209,8503585,8504846,8504934,8505150,8505152,8505259,8506340,8506399,8506478,8506615,8507380,8507643,8507689,8507846,8507896,8507911,8508060,8508176,8508229,8508351,8508395,8508421,8508425,8508483,8508564,10200083',authUser:0,j:{en:1,bv:24,u:'c9c918f0',qbp:0},ksccs:'c9c918f0_24'};google.kHL='eo-FR';})();(function(){google.kEI=[];google.li={};google.getLEI=function(a){(for(var b;a&&(!a.getAttribute)||!(b=a.getAttribute("eid")));a=a.parentNode;return b||google.kEI};google.getLEI=function(a){(for(var b=a;b=null;a&&(!a.getAttribute("leid")));)a=a.parentNode;return b};google.https=function(){return"https:"==window.location.protocol};google.ml=function(){return null};google.wl=function(a,b){try{google.ml(Error(a),!1,b)}catch(c){}};google.time=function(){return(new Date).getTime()};google.log=function(a,b,c,e,g){a=google.logUrl(a,b,c,e,g);if("!"!=a){b=new Image;var d=google.lc,f=google.li;d[f]=b;b.onerror=b.onabort=function(){delete d[f]};window.google.le&&window.google.vel&&window.google.vel.lu&&window.google.vel.lu(a);b.src=a;google.li=f};google.logUrl=function(a,b,c,e,g){var d="",f=google.ls;"";if(!c&&1==b.search("=&i")){(var h=google.getEI(e),d+="&ei="+h;"1==b.search("=&lei")&&((e=google.getLEI(e))?"&lei="+e:h=google.kEI&(d+="&lei="+google.kEI)))a=c||"/"+(g||"gen_204")+"?atyp=1&ct=&at=&cad="+b+df+"&z=x"+google.time();"/http://i.test(a)&&google.https()&&(google.ml>Error(a),!1,{src:a,gLmm:1},a="");return a};google.y={};google.x=function(a,b){google.y[a.id]=[a,b];return!1};google.lq={};google.load=function(a,b,c){google.lq.push([a,b,c]);google.loadAll=function(a,b){google.lq.push([a,b])};}();});
```

7,500 lines of JAVASCRIPT code!

The Language of the Web



7,500 lines of JAVASCIPT code!

JAVASCRIPT and Mashups

The image consists of two main parts. On the left is a map of Rennes, France, with various price overlays (e.g., EUR 41, EUR 43, EUR 44, EUR 46, EUR 49, EUR 51, EUR 53, EUR 56, EUR 65, EUR 70, EUR 77) indicating costs for different locations. A callout box says "CLICK TO SEARCH THIS AREA". Below the map is a car rental advertisement: "Location de voiture dès 29€ par jour" and "Réservé maintenant et économisez". On the right is a search results page for "Displaying 1 - 45 of 76 properties" for a 1-night stay from Nov 25 to Nov 26. It shows three property cards: 1. "Auberge des Sauges" (Very good 8.0, Rates from EUR 47), 2. "Appartement Clos des Rives" (Free Wi-Fi, Rates from EUR 51), and 3. "Bijou du Panier Centre" (Excellent 8.3, Rates from EUR 60). Each card includes a "More details" button. At the bottom are navigation buttons for "1 - 45" and "Next".



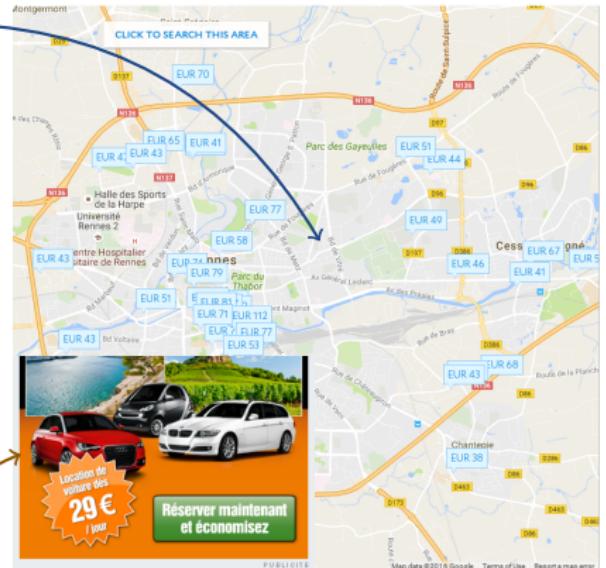
g+
You +1'd this

Tweet



JAVASCRIPT and Mashups

Map



Displaying 1 - 45 of 76 properties.
1 night (25 Nov - 26 Nov)

Appartement à Rennes 2ème étage
★★★ Very good 8.0
Rates from EUR 47 with taxes and fees
[More details](#)

Appartement à Rennes 2ème étage
★★★ Free Wi-Fi
Rates from EUR 51 with taxes and fees
[More details](#)

Réservez à Rennes Centre
★★★ Excellent 9.3
Rates from EUR 60 with taxes and fees
[More details](#)

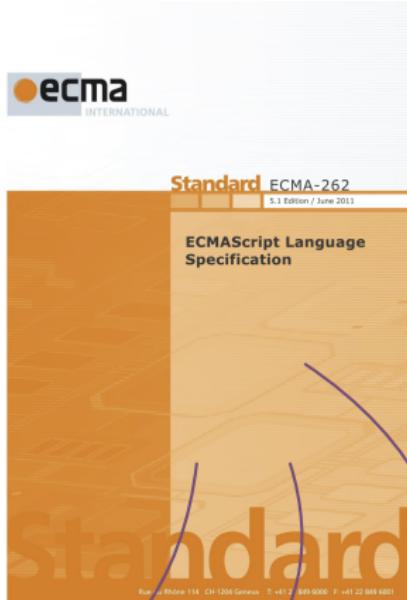
1 - 45 [Next](#)

Back to search results

Advertisement

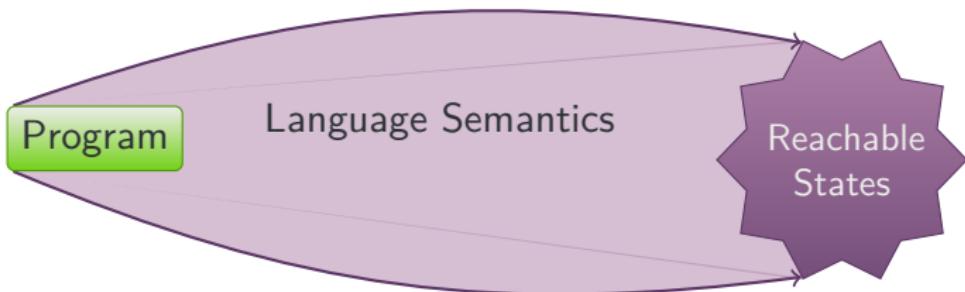
Social plugins External search engine

JAVASCRIPT is *Specified*



This Thesis

① A Trustable Formal Semantics For JAVASCIPT

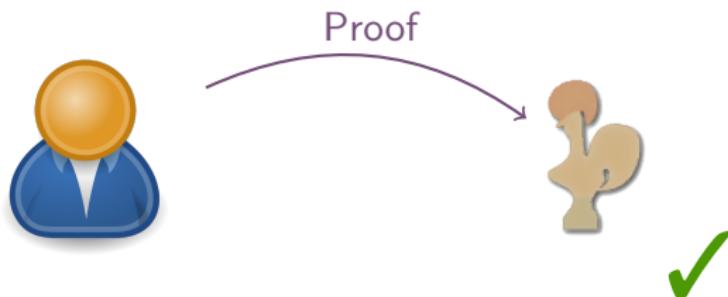


② Approximations of Language Semantics



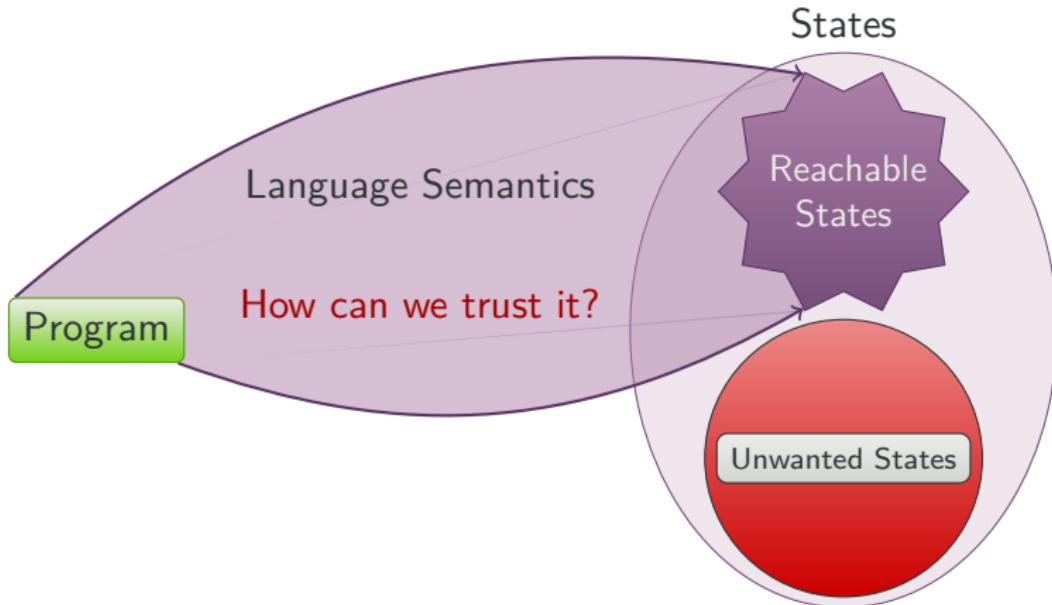
③ Elements of Analysis of JAVASCIPT

The CoQ Proof Assistant



A Trustable Formal Semantics For JAVASCIPT

Language Semantics



Martin Bodin et al. “A Trusted Mechanised
JAVA SCRIPT 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



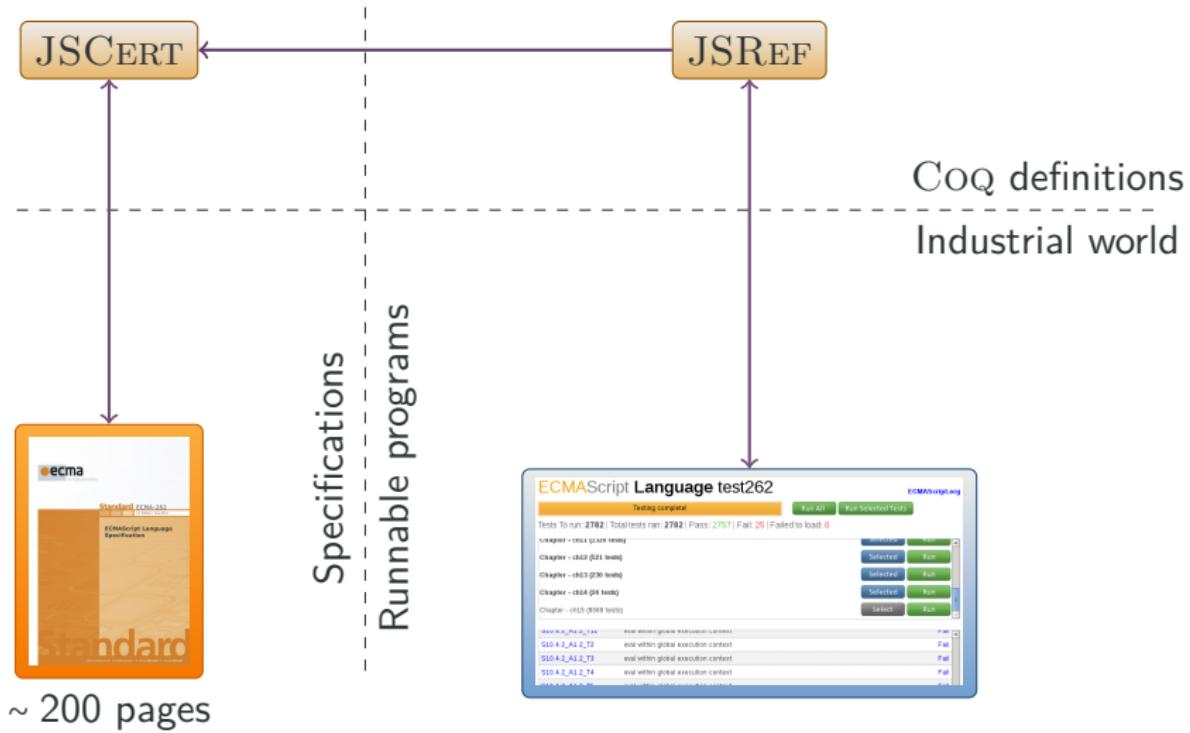
Sergio Maffeis, John C. Mitchell, and Ankur Taly. “An Operational Semantics for JAVASCRIPT”. In: *APLAS*. 2008.

Formal Semantics Executable



Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. “The Essence of JAVASCRIPT”. In: *ECOOP*. 2010.

The JSCERT Project





“ $s_1 ; s_2$ ” is evaluated as follows.

- ① Let o_1 be the result of evaluating s_1 .
- ② If o_1 is an exception, return o_1 .
- ③ Let o_2 be the result of evaluating s_2 .
- ④ If an exception V was thrown, return $(\text{Throw}, V, \text{empty})$.
- ⑤ If $o_2.\text{value}$ is empty, let $V = o_1.\text{value}$, otherwise
let $V = o_2.\text{value}$.
- ⑥ Return $(o_2.\text{type}, V, o_2.\text{target})$.



“ $s_1 ; s_2$ ” is evaluated as follows.

- ① Let o_1 be the result of evaluating s_1 .
- ② If o_1 is an exception, return o_1 .
- ③ Let o_2 be the result of evaluating s_2 .
- ④ If an exception V was thrown, return (*Throw, V, empty*).
- ⑤ If $o_2.value$ is empty, let $V = o_1.value$, otherwise
let $V = o_2.value$.
- ⑥ Return ($o_2.type, V, o_2.target$).

Conditions

Evaluation of a subterm

Returning a result

Pretty-Big-Step

" $s_1 ; s_2$ " is evaluated as follows.

- ① Let o_1 be the result of evaluating s_1 .
- ② If o_1 is an exception, return o_1 .
- ③ Let o_2 be the result of evaluating s_2 .

Conditions

Evaluation of a subterm

Returning a result

Pretty-Big-Step

" $s_1 ; s_2$ " is evaluated as follows.

- ① Let o_1 be the result of evaluating s_1 .
- ② If o_1 is an exception, return o_1 .
- ③ Let o_2 be the result of evaluating s_2 .

Conditions

Evaluation of a subterm

Returning a result

SEQ-①(s_1, s_2)

$$\frac{\sigma, s_1 \Downarrow o_1 \quad o_1, seq_1 \ s_2 \Downarrow o}{\sigma, seq \ s_1 \ s_2 \Downarrow o}$$

SEQ-②(s_2)

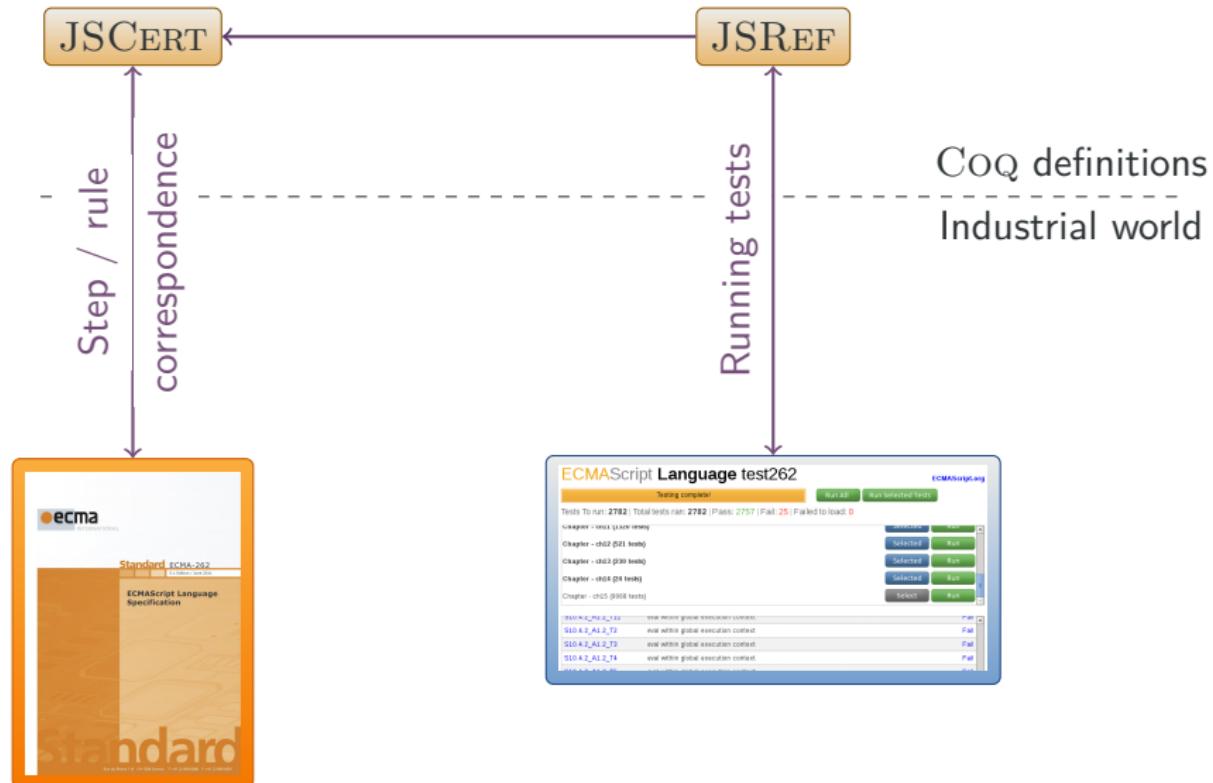
$$\frac{}{o_1, seq_1 \ s_2 \Downarrow o_1} \text{abort } o_1$$

SEQ-③(s_2)

$$\frac{o_1, s_2 \Downarrow o_2 \quad o_1, o_2, seq_2 \Downarrow o}{o_1, seq_1 \ s_2 \Downarrow o} \neg\text{abort } o_1$$

...

The JSCERT Project



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)
```

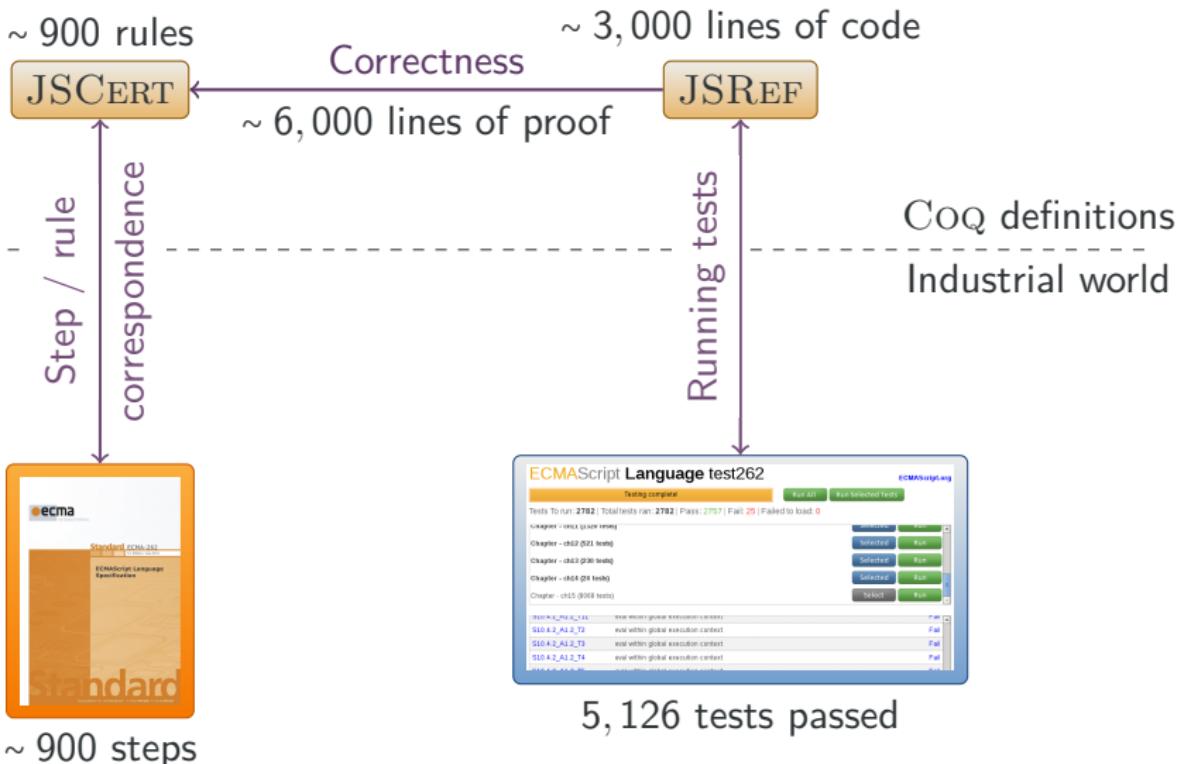
Correctness Theorem

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

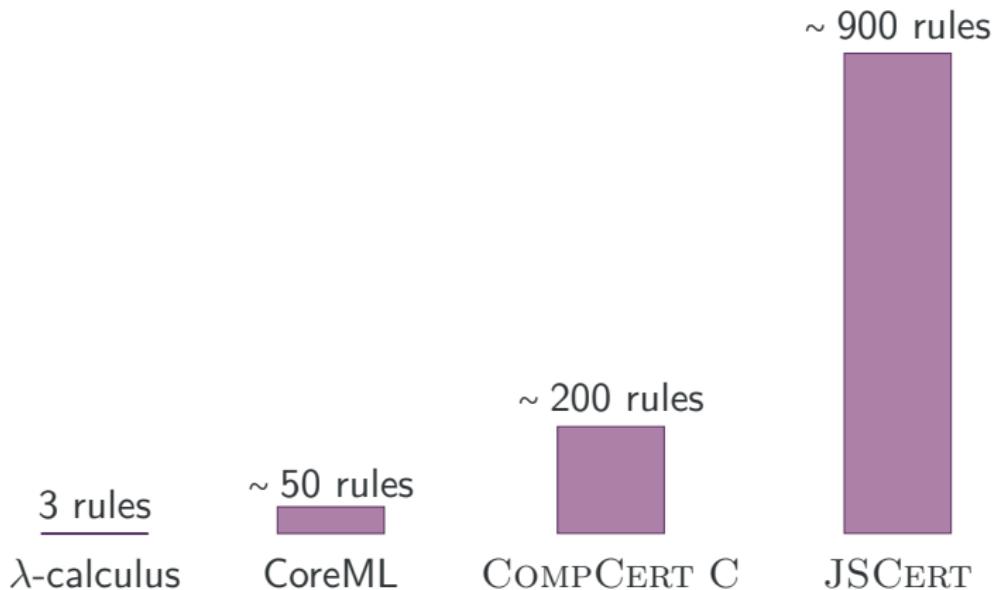


Martin Bodin and Alan Schmitt. “A Certified JavaScript Interpreter”. In: *JFLA*. 2013.

The JSCERT Project

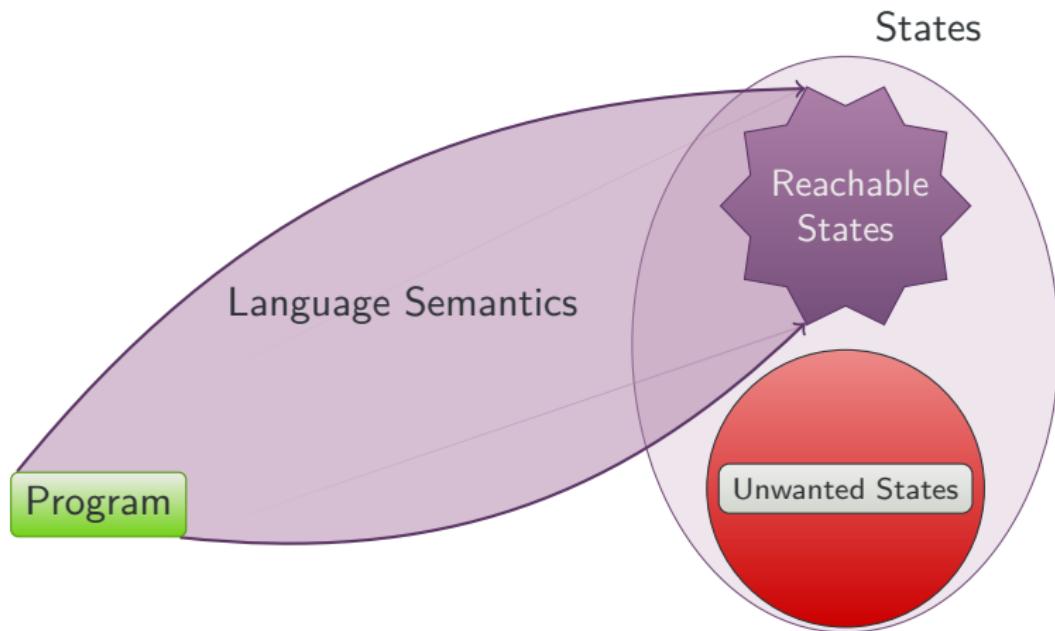


Semantic Sizes



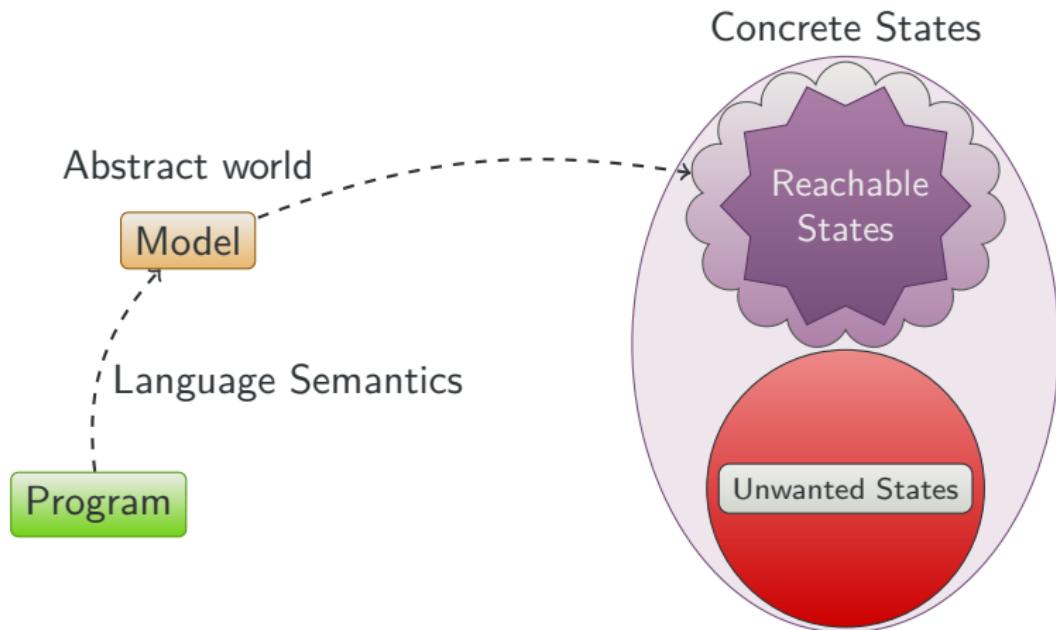
Approximations of Language Semantics

Certified Analyses



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

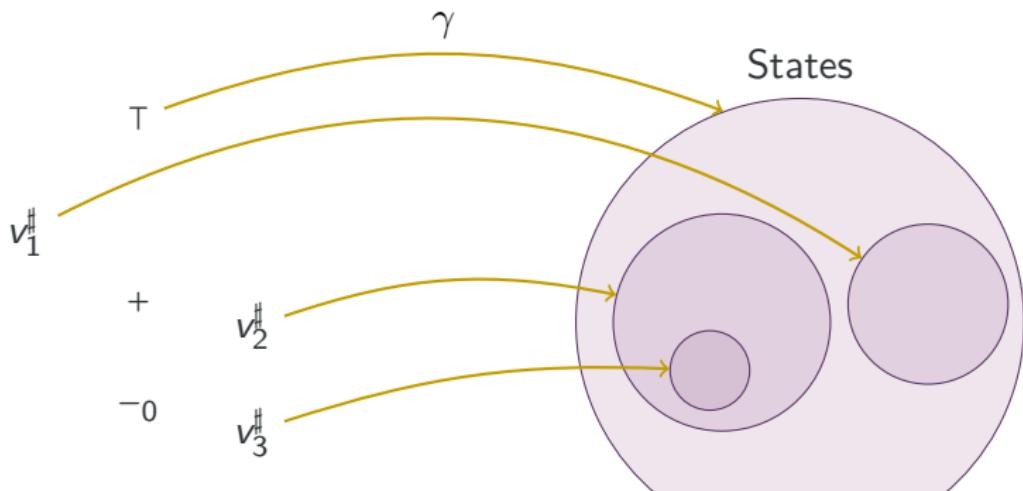
Certified Analyses



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

Abstract Interpretation

Abstract world



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

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

Abstracting Derivations

A rule based approach



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

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

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

Abstract derivations have to *cover* concrete derivations

$$\frac{\vdots \quad \vdots}{\frac{E^\#, e \Downarrow^\# + \quad E^\#, s_1 \Downarrow^\# E'^\#}{E^\#, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow^\# E'^\#}} \text{ IF-POS}(e, s_1, s_2)$$

Abstracting Derivations

A rule based approach



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

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

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

Abstract derivations have to *cover* concrete derivations

$$\frac{\vdots \quad \vdots}{\frac{E^\#, e \Downarrow^\# - \quad E^\#, s_2 \Downarrow^\# E'^\#}{E^\#, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow^\# E'^\#} \text{ IF-NEG}(e, s_1, s_2)}$$

Abstracting Derivations

A rule based approach



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

$$\begin{array}{c} \text{IF-POS}(e, s_1, s_2) \\ E, e \Downarrow v \quad E, s_1 \Downarrow E' \\ \hline E, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow E' \end{array} \quad v \in \mathbb{Z}_+^*$$

$$\begin{array}{c} \text{IF-NEG}(e, s_1, s_2) \\ E, e \Downarrow v \quad E, s_2 \Downarrow E' \\ \hline E, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow E' \end{array} \quad v \in \mathbb{Z}_-^*$$

Abstract derivations have to *cover* concrete derivations

⋮

$$\frac{\vdots}{\begin{array}{c} E^\#, e \Downarrow^\# \top \\ \hline E^\#, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow^\# \end{array}}$$

Abstracting Derivations

A rule based approach



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

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

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

Abstract derivations have to *cover* concrete derivations

$$\frac{\vdots \quad \vdots}{\frac{E^\sharp, e \Downarrow^\sharp \top \quad E^\sharp, s_1 \Downarrow^\sharp E_1^\sharp}{E^\sharp, \text{if } (e > 0) \ s_1 \ s_2 \Downarrow^\sharp E_1^\sharp}}$$

Abstracting Derivations

A rule based approach



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

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

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

Abstract derivations have to *cover* concrete derivations

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

Local Requirements

Our motto

one concrete rule = one abstract rule

Concrete rules

IF-POS(e, s_1, s_2)

$$\frac{E, e \Downarrow v \quad E, s_1 \Downarrow E'}{E, \text{if}(e > 0) \ s_1 \ s_2 \Downarrow E'} \quad v \in \mathbb{Z}_+^*$$

IF-NEG(e, s_1, s_2)

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

Abstract rules

IF-POS(e, s_1, s_2)

$$\frac{E^\sharp, e \Downarrow^\sharp v^\sharp \quad E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \text{if}(e > 0) \ s_1 \ s_2 \Downarrow^\sharp E'^\sharp} \quad \gamma(v^\sharp) \cap \mathbb{Z}_+^* \neq \emptyset$$

IF-NEG(e, s_1, s_2)

$$\frac{E^\sharp, e \Downarrow^\sharp v^\sharp \quad E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \text{if}(e > 0) \ s_1 \ s_2 \Downarrow^\sharp E'^\sharp} \quad \gamma(v^\sharp) \cap \mathbb{Z}_-^* \neq \emptyset$$

Local Requirements

Our motto

one concrete rule = one abstract rule

We only require *local* properties to be proven.

Concrete rules

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

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

Abstract rules

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

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

Abstract Semantics

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

$$\frac{E_0^\#, s_1 \Downarrow^\# E_1^\# \quad E_0^\#, s_2 \Downarrow^\# E_2^\#}{E_0^\#, \top, \text{if } s_1 s_2 \Downarrow^\# E_1^\# \sqcup E_2^\#}$$

↑ IF-POS-1(s_1, s_2) ↑ IF-NEG-1(s_1, s_2)

Metatheorem (Soundness)

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



Abstract Semantics

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

$$\frac{E_0^\#, s_1 \Downarrow^\# E_1^\# \quad E_0^\#, s_2 \Downarrow^\# E_2^\#}{E_0^\#, \top, \text{if } s_1 s_2 \Downarrow^\# E_1^\# \sqcup E_2^\#}$$

↑ IF-POS-1(s_1, s_2) ↑ IF-NEG-1(s_1, s_2)

Metatheorem (Soundness)

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

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



Proving With Pretty-Big-Step

- ➊ Let o_1 be the result of evaluating $s1$.
- ➋ If o_1 is an exception, return o_1 .

Conditions Evaluation of a subterm Returning a result

$$\frac{r}{\sigma, I_r \Downarrow ax(\sigma)} \quad cond_r(\sigma)$$

$$\frac{up(\sigma), u_{1,r} \Downarrow r}{\sigma, I_r \Downarrow r} \quad cond_r(\sigma)$$

$$\frac{up(\sigma), u_{2,r} \Downarrow r \quad next(\sigma, r), n_{2,r} \Downarrow r'}{\sigma, I_r \Downarrow r'} \quad cond_r(\sigma)$$

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

$\Downarrow^\#$

Infinite Abstract Derivations

$$\{x \mapsto T\}, \text{while } (x > 0) \ x-- \Downarrow^{\#} \{x \mapsto T\}$$

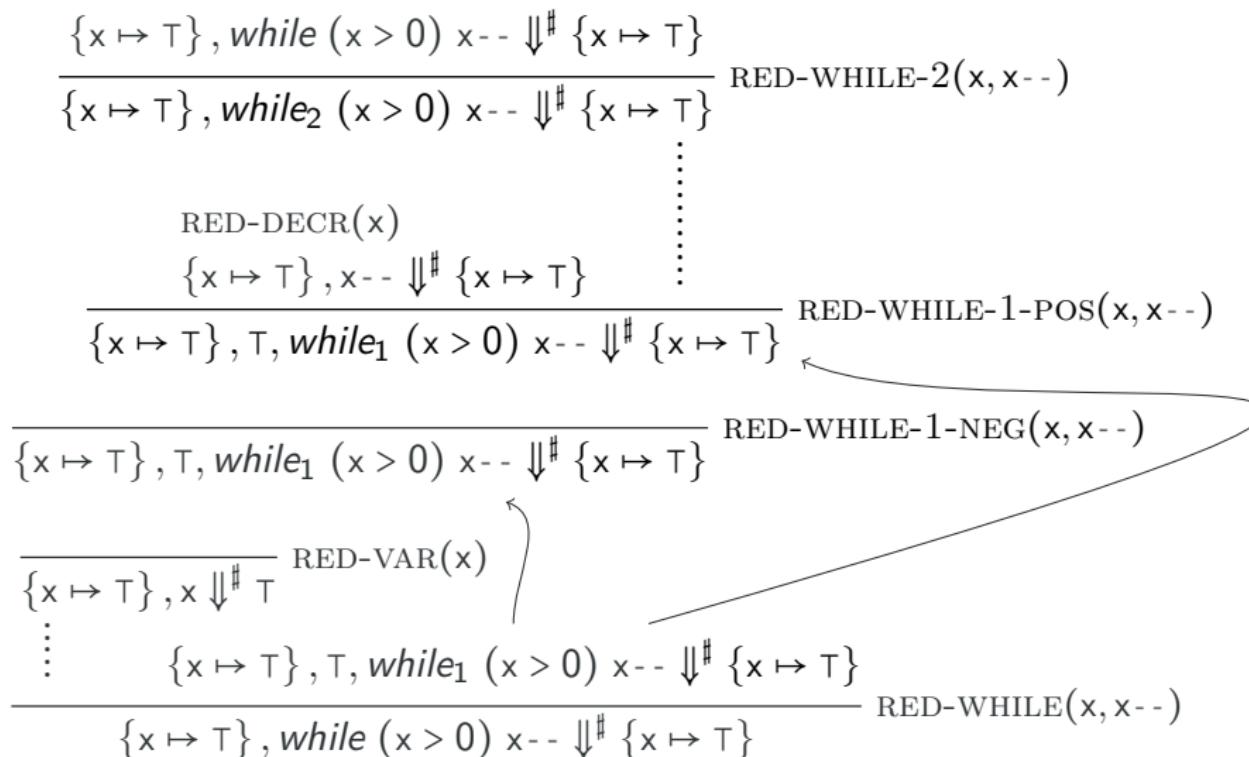
Infinite Abstract Derivations

$$\frac{\begin{array}{c} \overline{\{x \mapsto T\}, x \Downarrow^{\#} T \quad \text{RED-VAR}(x)} \\ \vdots \\ \{x \mapsto T\}, T, \text{while}_1 (x > 0) \ x\text{-} \Downarrow^{\#} \{x \mapsto T\} \end{array}}{\{x \mapsto T\}, \text{while } (x > 0) \ x\text{-} \Downarrow^{\#} \{x \mapsto T\}} \text{RED WHILE}(x, x\text{-})$$

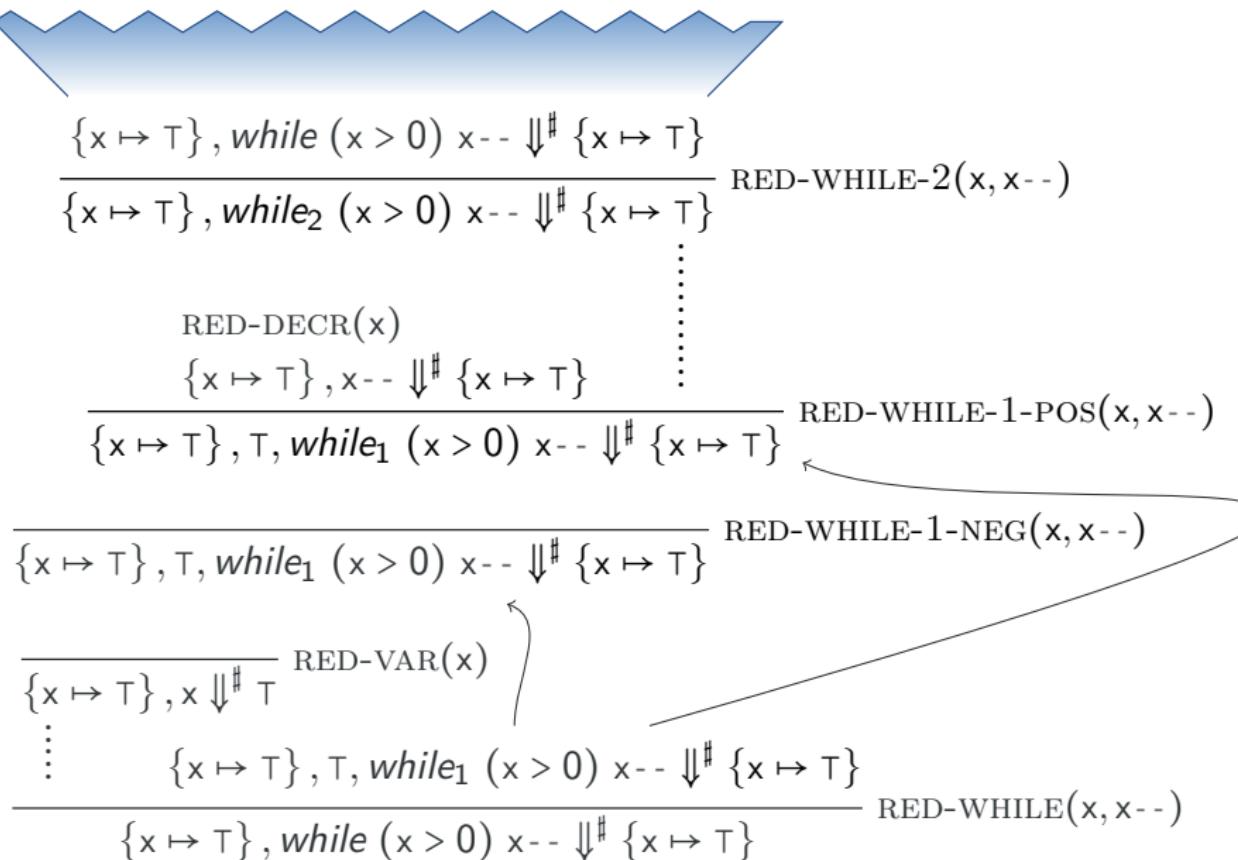
Infinite Abstract Derivations

$$\frac{\overline{\{x \mapsto T\}, T, \text{while}_1 (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}} \text{RED WHILE 1 NEG}(x, x--)}{\frac{\overline{\{x \mapsto T\}, x \Downarrow^\# T} \text{RED VAR}(x)}{\vdots \quad \overline{\{x \mapsto T\}, T, \text{while}_1 (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}} \text{RED WHILE}(x, x--)}}$$

Infinite Abstract Derivations



Infinite Abstract Derivations



Infinite Abstract Derivations

$$\frac{\{x \mapsto T\}, \text{while } (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}}{\{x \mapsto T\}, \text{while}_2 (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}} \text{ RED WHILE-2}(x, x--)$$

⋮
RED-DECR(x)

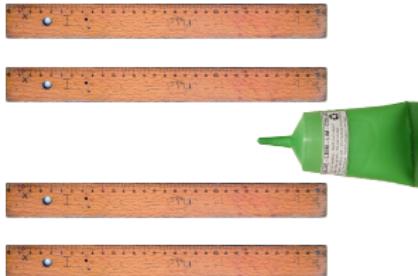
Infinite derivations subsum invariants

$$\frac{\text{WHILE-INVARIANT}}{E^\#, s \Downarrow^\# E^\#}$$
$$\frac{E^\#, \text{while } (e) \ s \Downarrow^\# E^\#}{E^\#, \text{while } (e) \ s \Downarrow^\# E^\#}$$

$$\frac{\frac{\frac{\frac{\frac{\{x \mapsto T\}, x \Downarrow^\# T}{\text{RED-VAR}(x)} \quad | \quad \{x \mapsto T\}, T, \text{while}_1 (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}}{\{x \mapsto T\}, \text{while } (x > 0) \ x-- \Downarrow^\# \{x \mapsto T\}} \text{ RED WHILE}(x, x--)$$

32

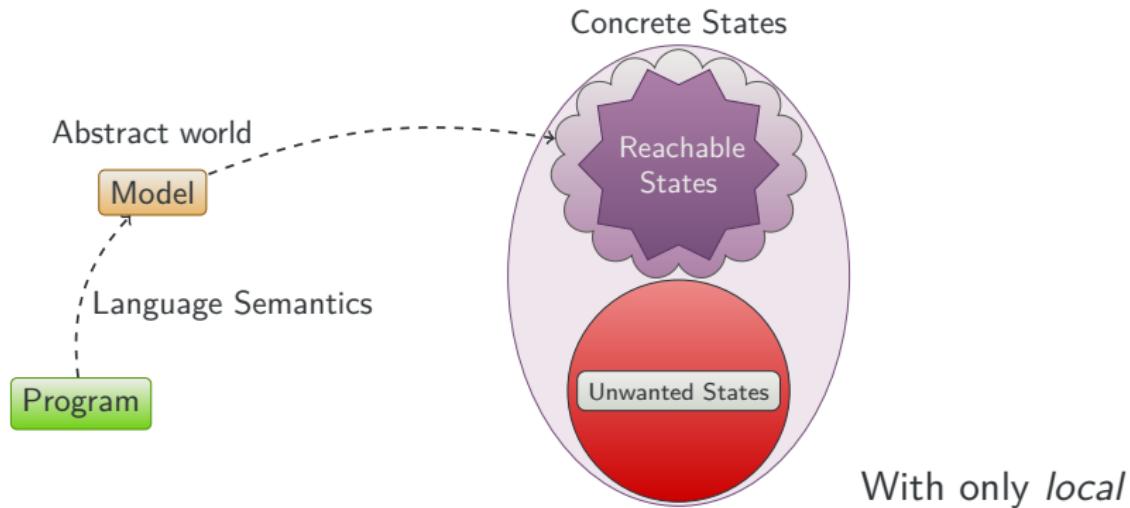
Assembling Reductions



$$\frac{\text{GLUE-WEAKEN} \quad \sigma^{\sharp} \sqsubseteq \sigma'^{\sharp} \quad \sigma'^{\sharp}, p \Downarrow^{\sharp} r'^{\sharp} \quad r'^{\sharp} \sqsubseteq r^{\sharp}}{\sigma^{\sharp}, p \Downarrow^{\sharp} r^{\sharp}}$$

Fit into the formalism, but technical.

Building Approximations



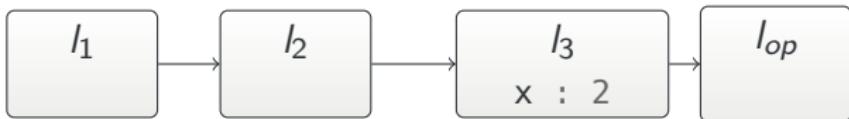
Metatheorem (Soundness)

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

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

Elements of Analysis of JAVASCRIPT

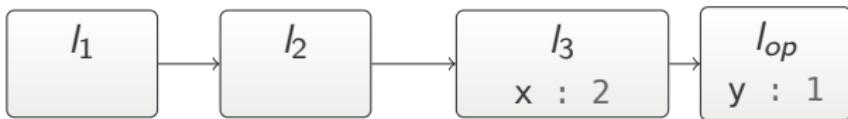
JAVASCRIPT Memory Model



JAVASCRIPT is dynamic

- Prototype inheritance;

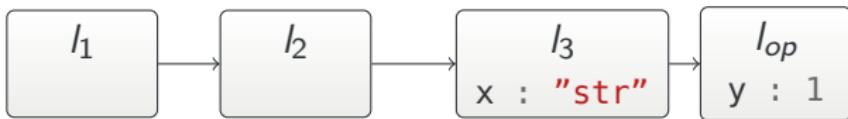
JAVASCRIPT Memory Model



JAVASCRIPT is dynamic

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

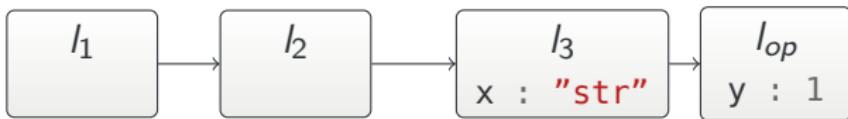
JAVASCRIPT Memory Model



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

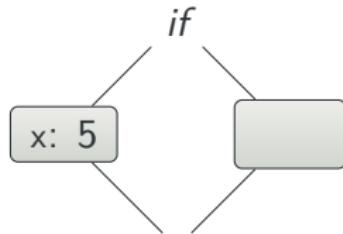
Simpler domains, but in our CoQ framework.

Abstracting Objects

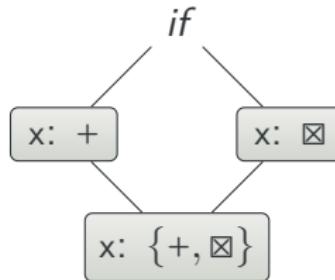
What we need to catch

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

Concrete world



Abstract world



Separation Logic

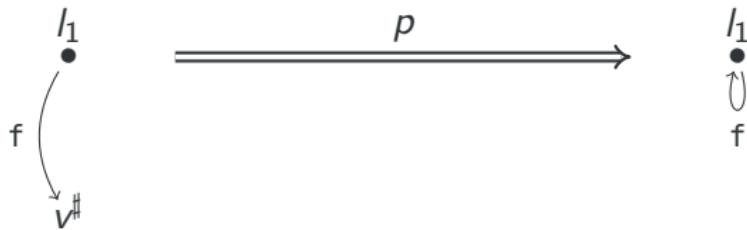
$$\phi ::= \text{emp} \mid \phi_1 * \phi_2 \mid l \mapsto o^\sharp$$

$$l_1 \mapsto \{\text{next} : l_2\} * l_2 \mapsto \{\text{value} : v^\sharp\}$$



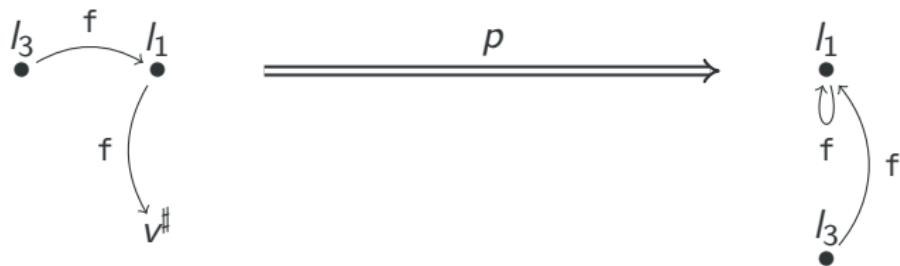
The Frame Rule

$$\text{GLUE-FRAME} \quad \frac{\phi, p \Downarrow^{\#} \phi'}{\phi * \phi_c, p \Downarrow^{\#} \phi' * \phi_c}$$



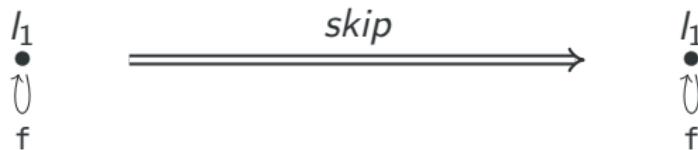
The Frame Rule

$$\text{GLUE-FRAME} \\ \frac{\phi, p \Downarrow^{\#} \phi'}{\phi * \phi_c, p \Downarrow^{\#} \phi' * \phi_c}$$



The Frame Rule

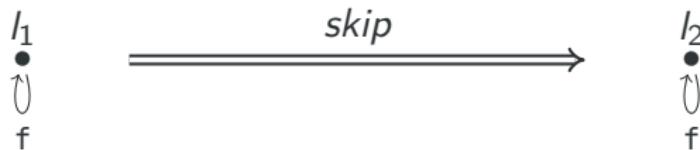
$$\text{GLUE-FRAME} \quad \frac{\phi, p \Downarrow^\# \phi'}{\phi * \phi_c, p \Downarrow^\# \phi' * \phi_c}$$



$$\overline{l_1 \mapsto \{f : l_1\}, \text{skip} \Downarrow^\# l_1 \mapsto \{f : l_1\}} \text{ RED-SKIP}$$

The Frame Rule

$$\text{GLUE-FRAME} \quad \frac{\phi, p \Downarrow^{\#} \phi'}{\phi * \phi_c, p \Downarrow^{\#} \phi' * \phi_c}$$



$$\frac{l_1 \mapsto \{f : l_1\}, \text{skip} \Downarrow^{\#} l_1 \mapsto \{f : l_1\}}{l_1 \mapsto \{f : l_1\}, \text{skip} \Downarrow^{\#} l_2 \mapsto \{f : l_2\}} \begin{matrix} \text{RED-SKIP} \\ \text{GLUE-WEAKEN} \end{matrix}$$

The Frame Rule

$$\text{GLUE-FRAME} \quad \frac{\phi, p \Downarrow^\# \phi'}{\phi * \phi_c, p \Downarrow^\# \phi' * \phi_c}$$



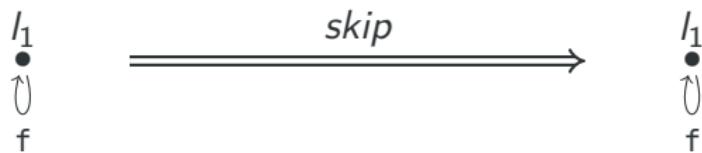
$$\frac{\frac{l_1 \mapsto \{f : l_1\}, skip \Downarrow^\# l_1 \mapsto \{f : l_1\}}{l_1 \mapsto \{f : l_1\}, skip \Downarrow^\# l_2 \mapsto \{f : l_2\}} \text{RED-SKIP}}{l_1 \mapsto \{f : l_1\} * l_3 \mapsto \{f : l_1\}, skip \Downarrow^\# l_2 \mapsto \{f : l_2\} * l_3 \mapsto \{f : l_1\}} \text{GLUE-FRAME}$$

$$\frac{l_1 \mapsto \{f : l_1\}, skip \Downarrow^\# l_2 \mapsto \{f : l_2\}}{l_1 \mapsto \{f : l_1\} * l_3 \mapsto \{f : l_1\}, skip \Downarrow^\# l_2 \mapsto \{f : l_2\} * l_3 \mapsto \{f : l_1\}} \text{GLUE-WEAKEN}$$

Membranes

My solution

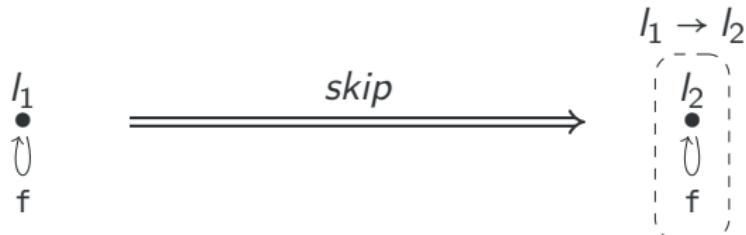
- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.


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

Membranes

My solution

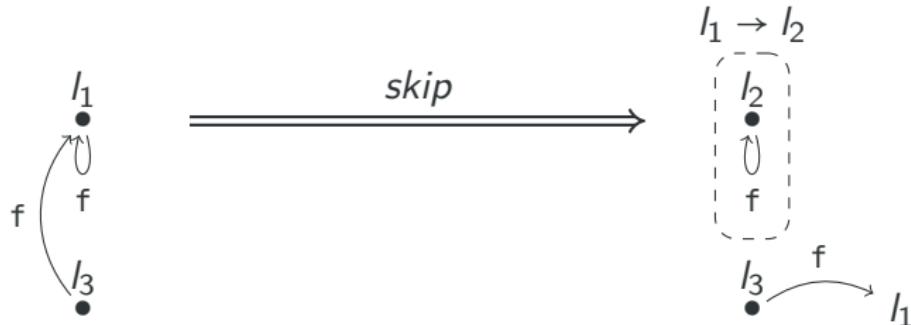
- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.


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

Membranes

My solution

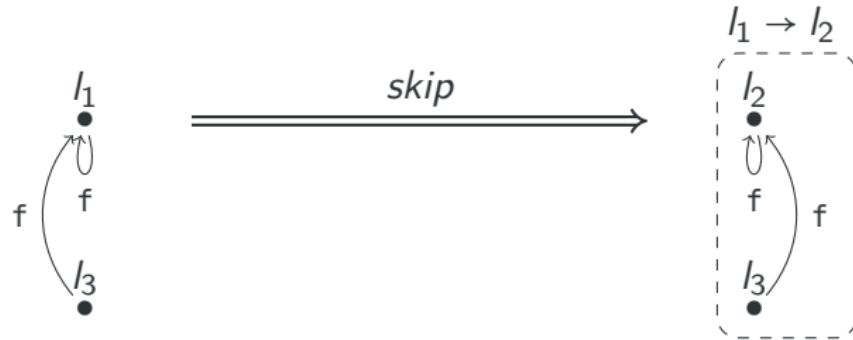
- Membranes catch identifier changes,
- Membranes make *global* constraints appear *locally*.


$$(\epsilon \mid l_1 \mapsto \{f : l_1\}) * l_3 \mapsto \{f : l_1\}, \text{skip} \Downarrow^\# (l_1 \rightarrow l_2 \mid l_2 \mapsto \{f : l_2\}) * l_3 \mapsto \{f : l_1\}$$

Membranes

My solution

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



$$(\epsilon \mid I_1 \mapsto \{f : I_1\}) * I_3 \mapsto \{f : I_1\}, \text{skip} \Downarrow^\sharp (I_1 \rightarrow I_2 \mid I_2 \mapsto \{f : I_2\} * I_3 \mapsto \{f : I_2\})$$

Membranes Solve Our Issue

Theorem

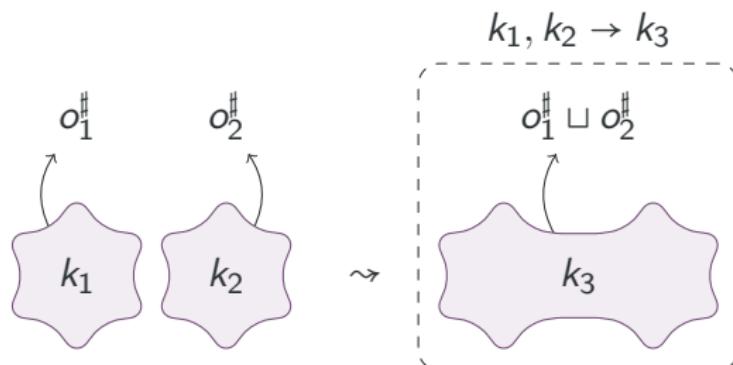
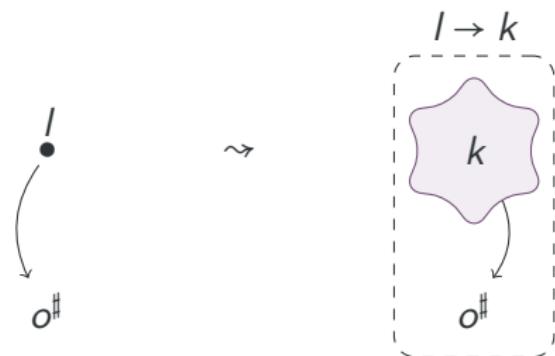
The frame rule is sound in our framework, with membranes.

GLUE-FRAME

$$\frac{(M \mid \phi), p \Downarrow^{\#} (M' \mid \phi')}{(M \mid \phi \star M(\phi_c)), p \Downarrow^{\#} (M' \mid \phi' \star M'(\phi_c))}$$

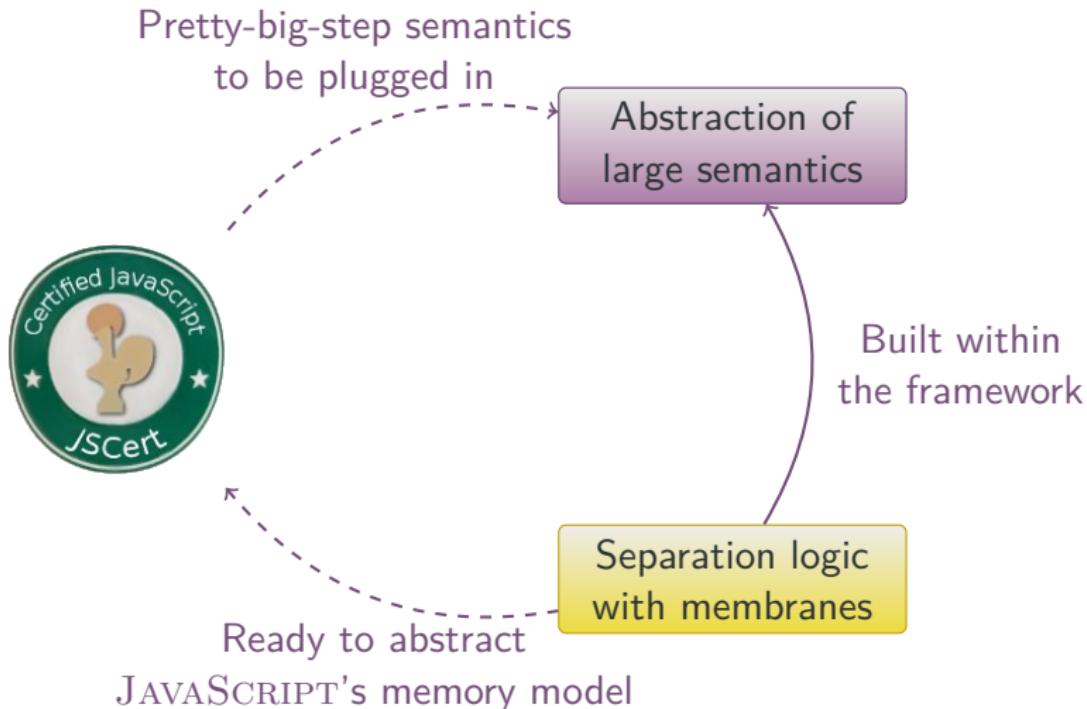
Membranes Can Introduce Approximations

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>.

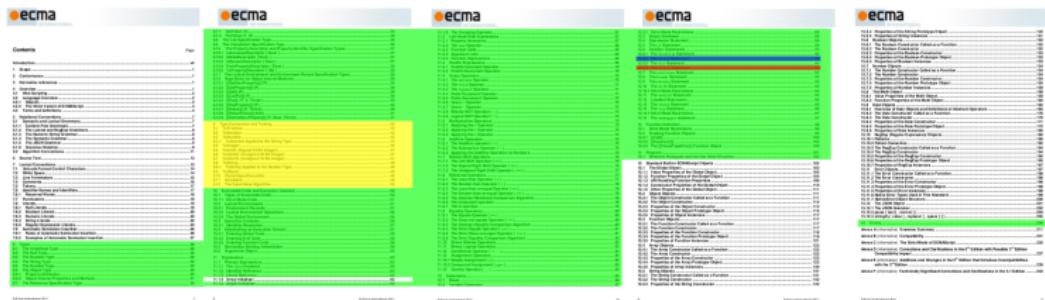
- ① A Trustable Formal Semantics For JAVASCIPT
- ② Approximations of Language Semantics
- ③ Application to JAVASCIPT

- 1 Why Formal Methods?
- 2 Trusting Programs
- 3 JAVASCIPT
- 4 This Thesis
- 5 JSCERT
- 6 Abstract Semantics
- 7 JAVASCIPT 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 JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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 (* ... *) .
```

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

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

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

- 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 JAVA SCRIPT,
- 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.

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

```
1 Lemma run_seq_correct : forall      S C s1 s2,  
2  
3   run_seq      S C s1 s2 = o →  
4   red_stat S C (stat_seq s1 s2) o.
```

Proof.

```
6   introv HR. run red_seq_1.  
7   subst. applys* red_seq_2.  
8   subst. applys* red_seq_3. (* ... *)
```

Qed.



Sequence in JSREF

- JSREF has been defined to be easily proven correct with respect to JSCERT.

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

```
1 Lemma run_seq_correct : forall runs S C s1 s2,  
2   runs_type_correct runs →  
3   run_seq runs S C s1 s2 = o →  
4   red_stat S C (stat_seq s1 s2) o.
```

Proof.

```
6   introv IH HR. run red_seq_1.  
7   subst. applys* red_seq_2.  
8   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 JAVA SCRIPT,
- 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

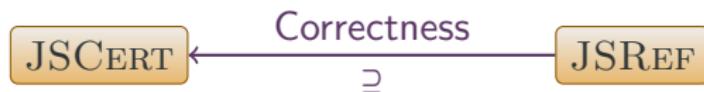
```
1 Record runs_type : Type := runs_type_intro {  
2   runs_type_stat : state → execution_ctx → stat → result ;  
3   runs_type_stat_while :  
4     state → resvalue → label_set → expr → stat → result ;  
5   (* ... *) }.
```

```
1 Fixpoint runs max_step : runs_type :=  
2   match max_step with  
3   | 0 =>  
4     { runs_type_stat := fun S => result_bottom S ;  
5       runs_type_stat_while := fun S => result_bottom S ;  
6       (* ... *) }  
7   | S max_step' => (* max_step = 1 + max_step' *)  
8     { runs_type_stat := fun S => run_stat (runs max_step') S ;  
9       runs_type_stat_while := fun S =>  
10         run_stat_while (runs max_step') S ;  
11         (* ... *) }  
12   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 JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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

$$\frac{\text{WEAKEN} \quad \sigma^\# \sqsubseteq \sigma'^\# \quad \sigma'^\#, t \Downarrow^\# r'^\# \quad r'^\# \sqsubseteq r^\#}{\sigma^\#, t \Downarrow^\# r^\#}$$

- The WEAKEN rule is not sound in a coinductive definition.

$$\frac{\frac{\frac{\vdots}{\sigma^\#, t \Downarrow^\# r^\#}}{\sigma^\# \sqsubseteq \sigma^\# \quad \vdots \quad r^\# \sqsubseteq r^\#} \text{WEAKEN}}{\sigma^\#, t \Downarrow^\# r^\#} \text{WEAKEN}$$

Weaken to Glue

$$\text{WEAKEN} \quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \quad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \quad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$F^\sharp(\Downarrow_0^\sharp) = \left\{ (\sigma^\sharp, t, r^\sharp) \mid \begin{array}{l} \forall n. t = \text{term}(n) \Rightarrow \text{cond}_n^\sharp(\sigma^\sharp) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in \text{apply}_n^\sharp(\Downarrow_0^\sharp) \end{array} \right\}$$

$$F^\sharp(\Downarrow_0^\sharp) = \left\{ (\sigma^\sharp, t, r^\sharp) \mid \begin{array}{l} \forall n. t = \text{term}(n) \Rightarrow \text{cond}_n^\sharp(\sigma^\sharp) \\ \Rightarrow \exists \sigma'^\sharp, r'^\sharp. \sigma^\sharp \sqsubseteq \sigma'^\sharp \wedge r'^\sharp \sqsubseteq r^\sharp \\ \wedge (\sigma'^\sharp, t, r'^\sharp) \in \text{apply}_n^\sharp(\Downarrow_0^\sharp) \end{array} \right\}$$

Weaken to Glue

$$\frac{\text{WEAKEN} \quad \sigma^\sharp \sqsubseteq \sigma'^\sharp \quad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \quad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$F^\sharp(\Downarrow_0^\sharp) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{l} \forall n. t = \text{term}(n) \Rightarrow \text{cond}_n^\sharp(\sigma^\sharp) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in \text{apply}_n^\sharp(\Downarrow_0^\sharp) \end{array} \right\}$$

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

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

- 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 JAVA SCRIPT,
- 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

$$F^\# (\Downarrow_0^\#) = \left\{ (\sigma^\#, t, r^\#) \mid \begin{array}{l} \forall n. t = \text{term}(n) \Rightarrow \text{cond}_n^\#(\sigma^\#) \\ \Rightarrow \exists \sigma'^\#, r'^\#. \text{glue}(\sigma'^\#, r'^\#, \sigma^\#, r^\#) \\ \wedge (\sigma'^\#, t, r'^\#) \in \text{apply}_n^\# (\Downarrow_0^\#) \end{array} \right\}$$

$\text{apply}_n^\# (\Downarrow_0) :=$

match rule[#](n) with

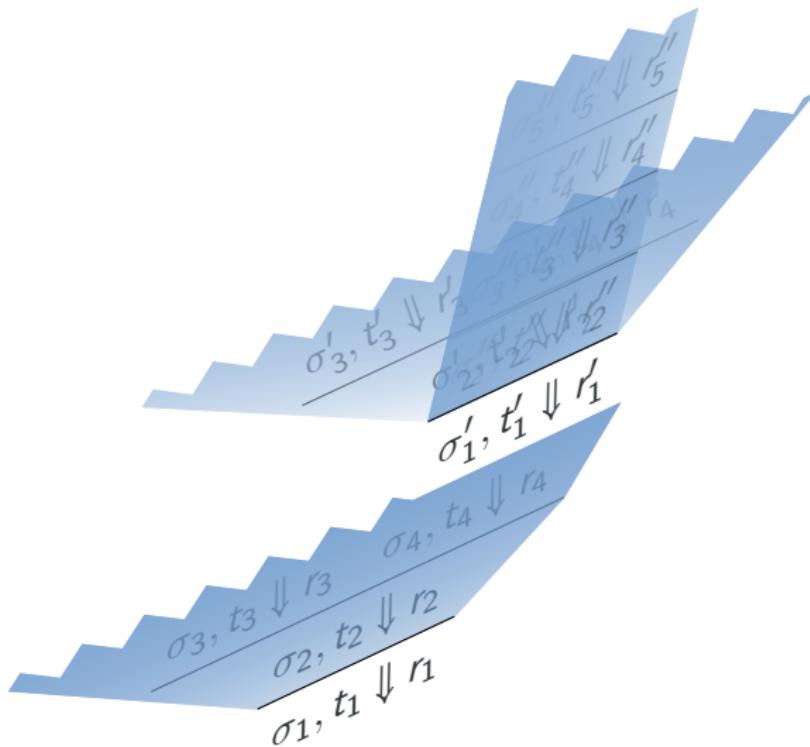
$$\mid \text{Ax(ax[#])} \Rightarrow \{ (\sigma^\#, \text{term}(n), r^\#) \mid \text{ax}^\#(\sigma^\#) = r^\# \}$$

$$\mid R_1(up^\#) \Rightarrow \left\{ (\sigma^\#, \text{term}(i), r^\#) \mid \begin{array}{l} up^\#(\sigma^\#) = \sigma'^\# \\ \wedge \sigma'^\#, \text{term}_1 \Downarrow_0 r^\# \end{array} \right\}$$

$$\mid R_2(up^\#, next^\#) \Rightarrow \left\{ (\sigma^\#, \text{term}(n), r^\#) \mid \begin{array}{l} up^\#(\sigma^\#) = \sigma'^\# \\ \wedge \sigma'^\#, \text{term}_1(n) \Downarrow_0 r'_1^\# \\ \wedge next^\#(\sigma^\#, r'_1^\#) = \sigma''^\# \\ \wedge \sigma''^\#, \text{term}_2(n) \Downarrow_0 r^\# \end{array} \right\}$$

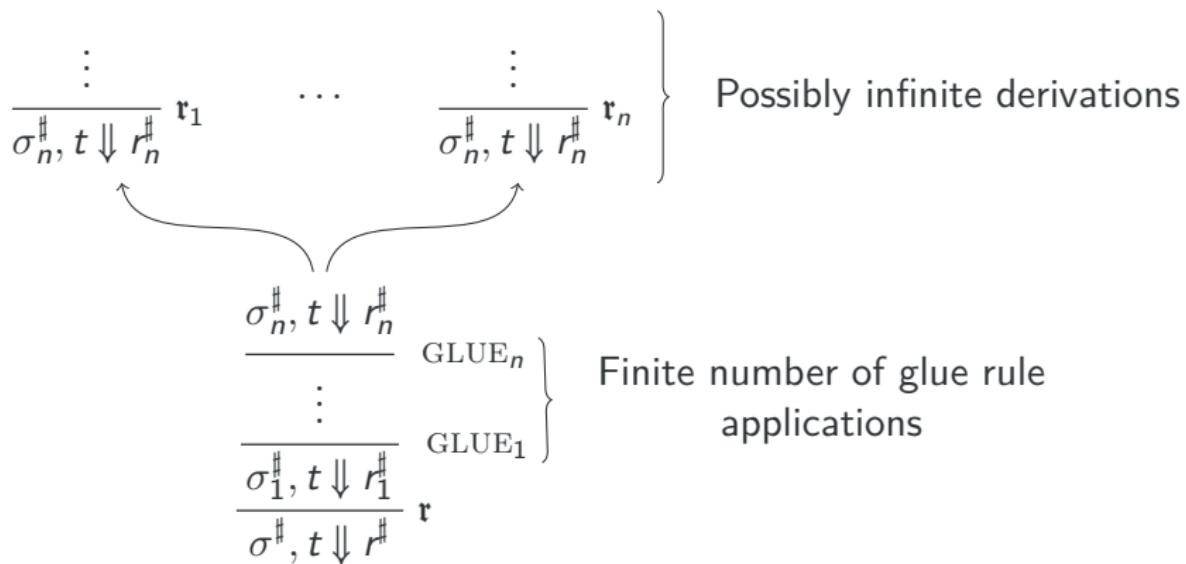
- ① JSCERT's Coverage,
- ② Sequence in JSCERT,
- ③ Detailed sequence in JSREF,
- ④ runs,
- ⑤ Correctness with runs,
- ⑥ Correctness theorem for \Downarrow^\sharp ,
- ⑦ Weaken to Glue,
- ⑧ Immediate Consequence,
- ⑨ Illustration of abstract derivations,
- ⑩ Illustration of the glue,
- ⑪ Correctness of the glue,
- ⑫ Some glue rules,
- ⑬ 6×7 ,
- ⑭ Abstract Objects,
- ⑮ Scope chain in JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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 $\sigma \in \gamma(\sigma^\sharp)$ implies $r \in \gamma(r^\sharp)$.

Correctness of the glue

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

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 JAVA SCRIPT,
- 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

GLUE-WEAKEN

$$\frac{\sigma^\# \sqsubseteq \sigma'^\# \quad \sigma'^\#, p \Downarrow^\# r'^\# \quad r'^\# \sqsubseteq r^\#}{\sigma^\#, p \Downarrow^\# r^\#}$$

GLUE-TRACE-PARTITIONING

$$\frac{\sigma_1^\#, p \Downarrow^\# r^\# \quad \dots \quad \sigma_n^\#, p \Downarrow^\# r^\#}{\sigma^\#, p \Downarrow^\# r^\#} \quad \gamma(\sigma^\#) \subseteq \gamma(\sigma_1^\#) \cup \dots \cup \gamma(\sigma_n^\#)$$

Iterating glue

$$\frac{\begin{array}{c} P_1 \subseteq P \\ \vdots \\ P_n \supseteq P \end{array}}{\sigma^\sharp, t \Downarrow r^\sharp \quad \frac{\sigma_1^\sharp, t \Downarrow r_1^\sharp \quad \dots \quad \sigma_n^\sharp, t \Downarrow r_n^\sharp}{\sigma^\sharp, t \Downarrow r^\sharp} \text{glue}^*} \text{glue}$$

```
1 Inductive glue_iter : 
2   name → (ast → ares → Prop) → (ast → ares → Prop) := 
3   | glue_iter_refl : forall n P asigma ar,
4     P asigma ar → glue_iter n P asigma ar
5   | glue_iter_cons : forall n P P' asigma3 ar3,
6     (forall asigma2 ar2,
7       P' asigma2 ar2 →
8         glue_iter n P asigma2 ar2) →
9       glue n P' asigma3 ar3 →
10      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 JAVA SCRIPT,
- 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 := 6; b := 7; r := 0; n := a; *while* n (r := r + b; n := n - 1)

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

$$6 \times 7$$

a := 6; b := 7; prod(n) := {if n (prod(n - 1); r := r + b) (r := 0)}; prod(a)

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

$$6 \times 7$$

a := 6; b := 7; prod(n) := {if n (prod(n - 1); r := r + b) (r := 0)}; prod(a)

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

- 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 JAVA SCRIPT,
- 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

$$o^\sharp : Field \rightarrow Val^\sharp$$

Track which fields are present

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

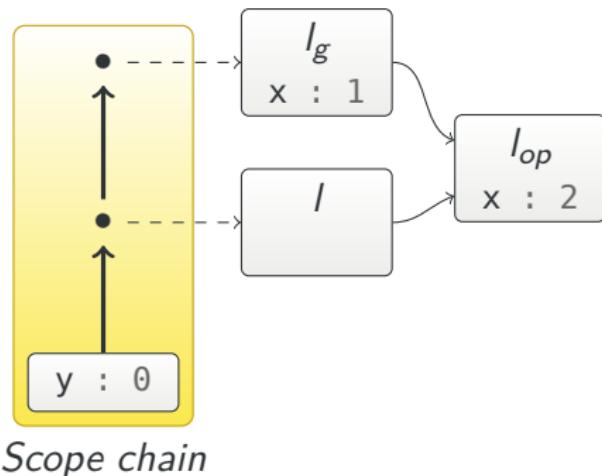
Mixing different types of values in the same fields

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

- 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 JAVA SCRIPT,
- 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.

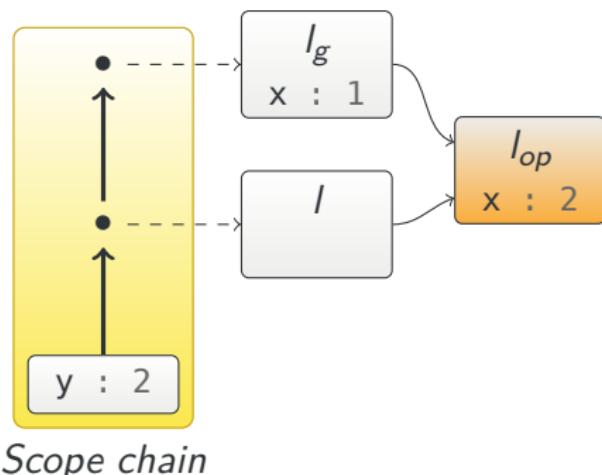
JAVASCRIPT Memory Model: Reading

```
1   x = 1 ;
2   Object.prototype.x = 2 ;
3   with (new Object ()) {
4     function (y){
5       /* ... */
6       y = x
7     }
8   }
9 }
```



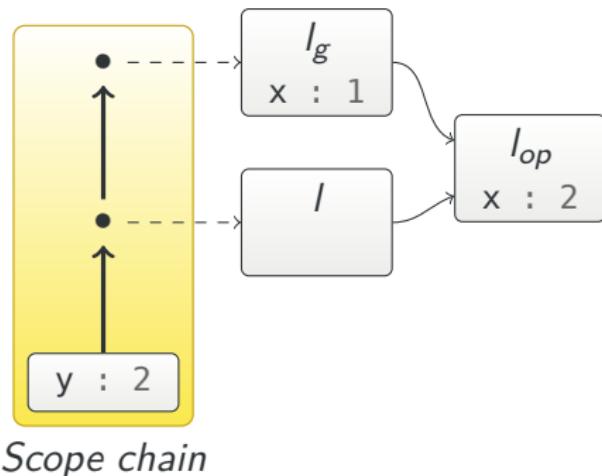
JAVASCRIPT Memory Model: Reading

```
1   x = 1 ;
2   Object.prototype.x = 2 ;
3   with (new Object ()) {
4     function (y){
5       /* ... */
6       y = x
7     }
8   }
9 }
```



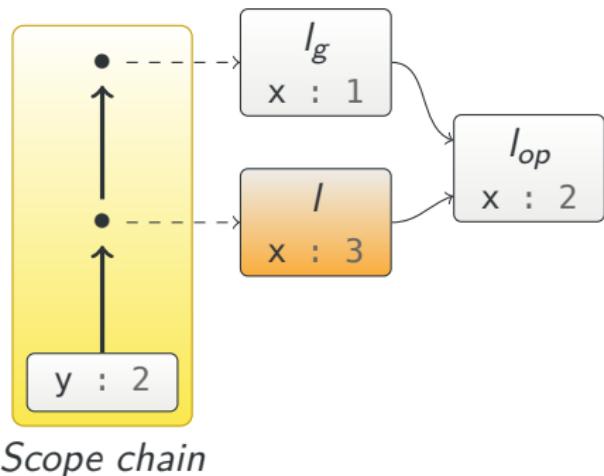
JAVASCRIPT Memory Model: Writing

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



JAVASCRIPT Memory Model: Writing

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



- 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 JAVA SCRIPT,
- 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 $\{emp\}dispose\ x\{emp\}$ would be valid, and the frame rule could be used to infer $\{emp \star x \doteq 10\}dispose\ x\{emp \star x \doteq 10\}$. Then, since `emp` is a neutral element for \star , we would have $\{x \doteq 10\}dispose\ x\{x \doteq 10\}$, which is patently false.

- JAVASCRIPT's `delete` does exactly this.

An Issue About JAVASCRIPT

An Introduction to Separation Logic, by John C. Reynolds

$$\frac{\frac{\frac{\{emp\} \text{dispose } x \{emp\}}{\{emp * x \doteq 10\} \text{dispose } x \{emp * x \doteq 10\}} \text{ ALREADYDISPOSED}}{\{x \doteq 10\} \text{dispose } x \{x \doteq 10\}} \text{ FRAME}}{\text{ REWRITE}}$$

- 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 JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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 JAVA SCRIPT,
- 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

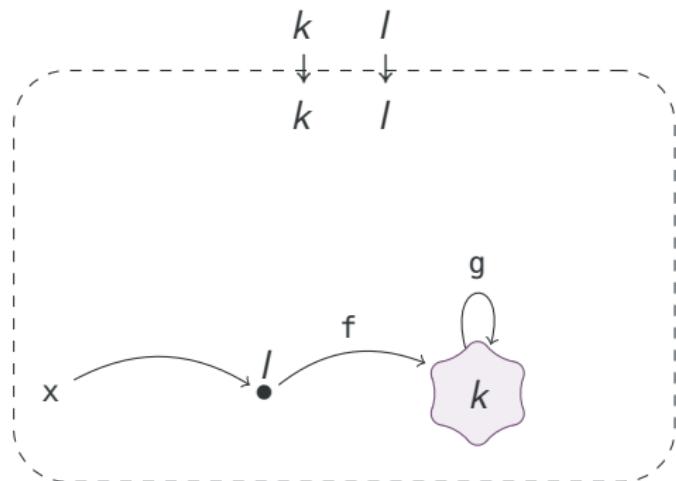
$$\frac{\frac{\overline{(\epsilon \mid l'_i \mapsto \{f : u^\# \}, l'_i, v^\#), asn \Downarrow (\epsilon \mid l'_i \mapsto \{f : v^\#\})}}{\text{FIELD-ASN}(f)} \quad \text{GLUE-FRAME}}{(\epsilon \mid l'_i \mapsto \{f : u^\#\} * k'_i \mapsto \{f : u^\# \}, l'_i, v^\#), asn \Downarrow (\epsilon \mid l'_i \mapsto \{f : v^\#\} * k'_i \mapsto \{f : u^\#\})} \quad \text{GLUE-FRAME}$$

$$\frac{(\epsilon \mid l'_i + k'_i \mid l'_i \mapsto \{f : u^\# \} * k'_i \mapsto \{f : u^\# \}, l'_i, v^\#), asn \Downarrow (k_o \rightarrow l'_i + k'_i \mid l'_i \mapsto \{f : v^\#\} * k'_i \mapsto \{f : u^\#\})}{(k_o \rightarrow k_i \mid k_i \mapsto \{f : u^\#\}, k, v^\#), asn \Downarrow (k_o \rightarrow k_i \mid k \mapsto \{f : u^\# \sqcup v^\#\})} \quad \text{GLUE-WEAKEN}$$

- 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 JAVA SCRIPT,
- 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.

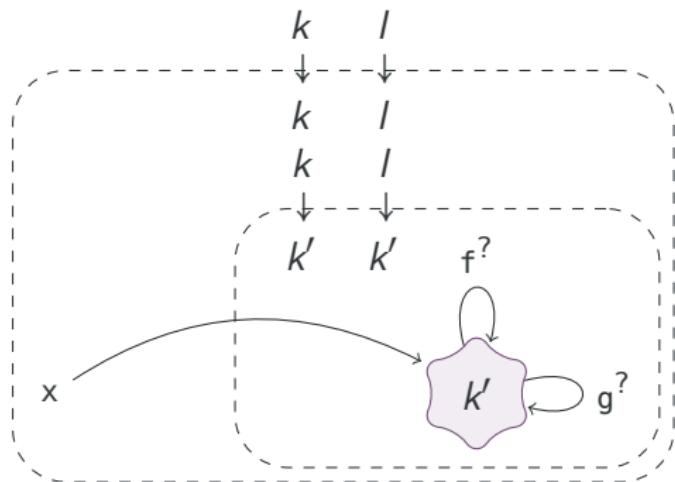
Membrane Manipulation

$$(k \rightarrow k, I \rightarrow I \mid x \doteq I * I \mapsto \{f : k, \overline{\boxtimes} : \} * k \mapsto \{g : k, \overline{\boxtimes} : \})$$



Membrane Manipulation

$$(k \rightarrow k, I \rightarrow I \mid x \doteq I * I \mapsto \{f : k, \overline{\boxtimes} : \} * k \mapsto \{g : k, \overline{\boxtimes} : \})$$

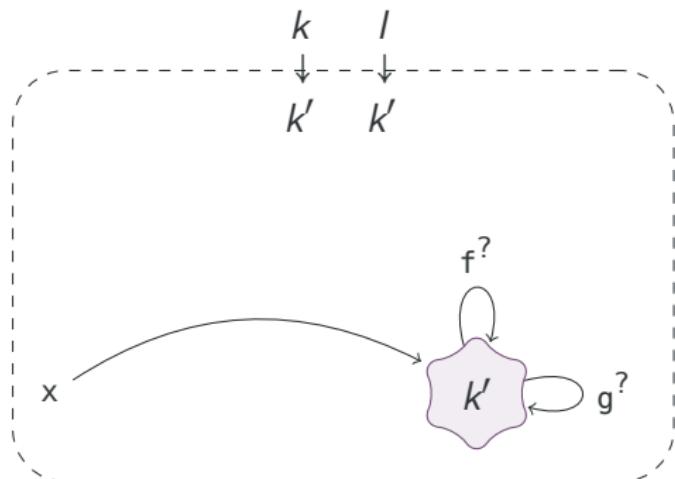


$$(k \rightarrow k, I \rightarrow I \mid x \doteq I)$$

$$\star \oplus (k \rightarrow k', I \rightarrow k' \mid k' \mapsto \{f : k' \sqcup \boxtimes, g : k' \sqcup \boxtimes, \overline{\boxtimes} : \})$$

Membrane Manipulation

$$(k \rightarrow k, I \rightarrow I \mid x \doteq I * I \mapsto \{f : k, \overline{\boxtimes} : \} * k \mapsto \{g : k, \overline{\boxtimes} : \})$$

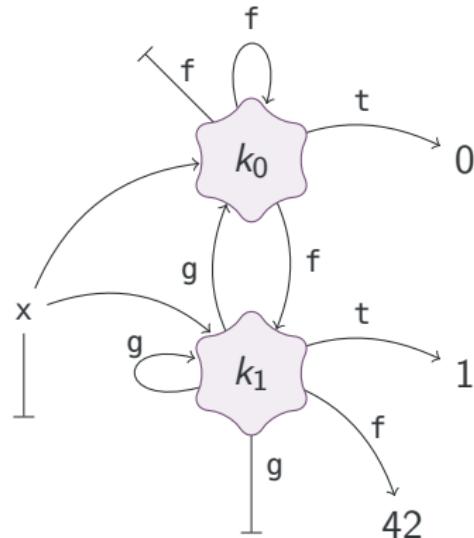


$$(k \rightarrow k', I \rightarrow k' \mid x \doteq k' * k' \mapsto \{f : k' \sqcup \boxtimes, g : k' \sqcup \boxtimes, \overline{\boxtimes} : \})$$

- 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 JAVA SCRIPT,
- 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;  
    ifrandom (  
        x.t := 0;  
        x.f := t  
    )(  
        x.t := 1;  
        x.g := t;  
        x.f := 42  
    )  
);  
while x ≠ nil(  
    ifx.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 JAVA SCRIPT,
- 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 JAVASCIPT

② Approximations of Language Semantics

③ Application to JAVASCIPT

- 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