Towards Tractable Verification of JavaScript Programs

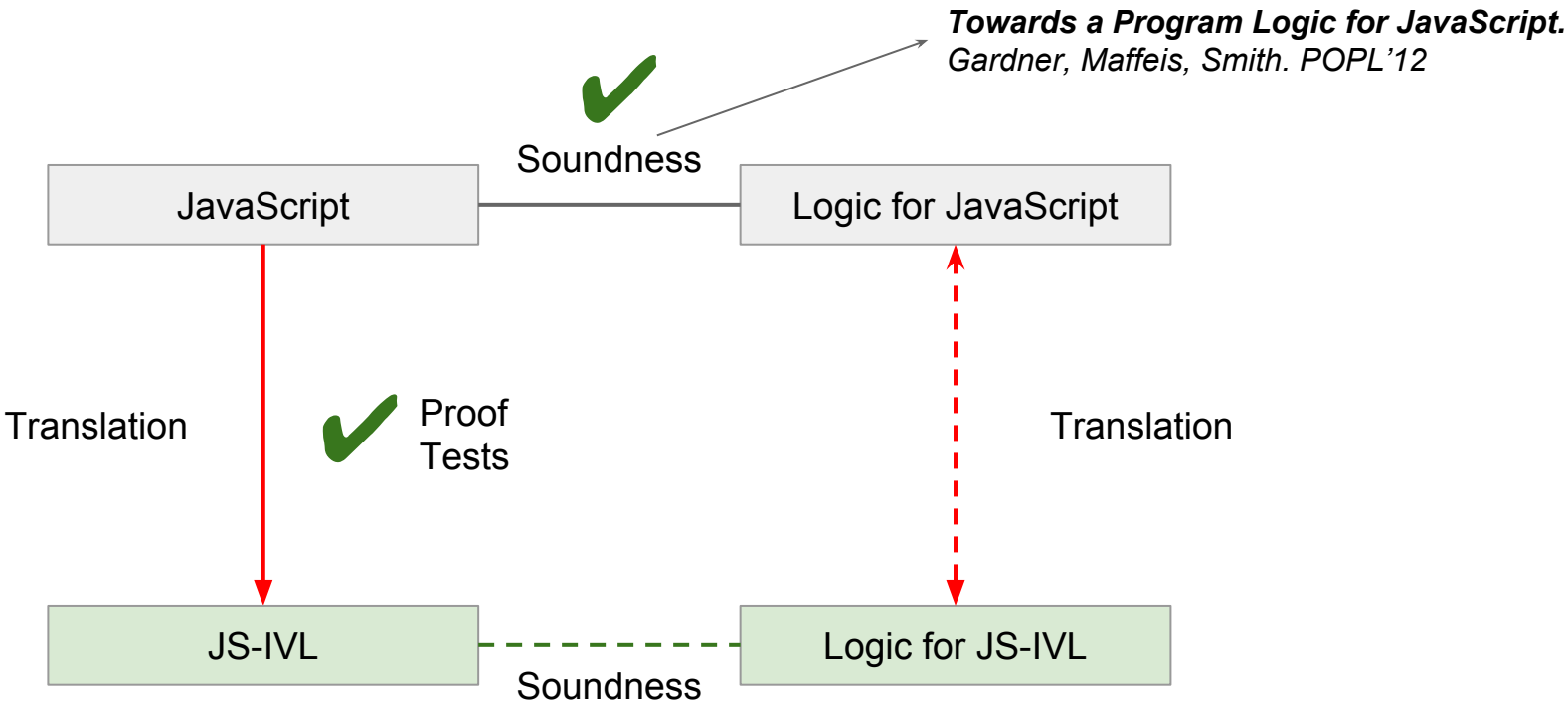
Philippa Gardner, **Daiva Naudžiūnienė**, José Fragoso Santos
Imperial College London

JSTools 2015

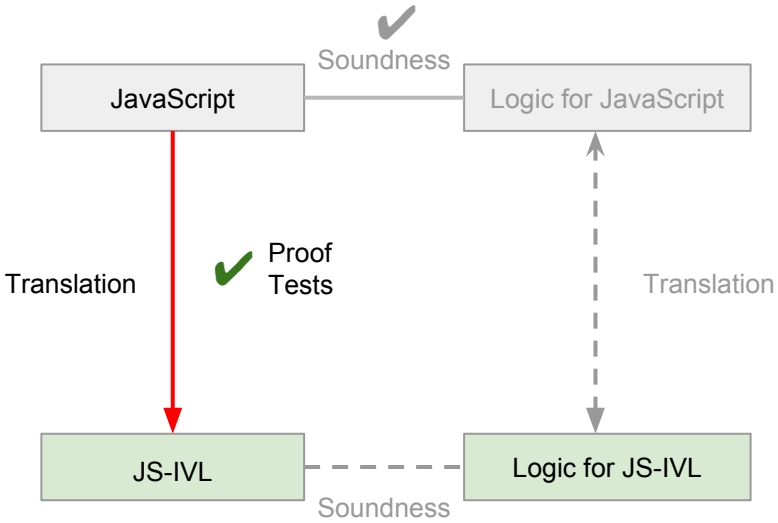
The Goal

Verification Tool (based on Separation Logic) for JavaScript

The Approach



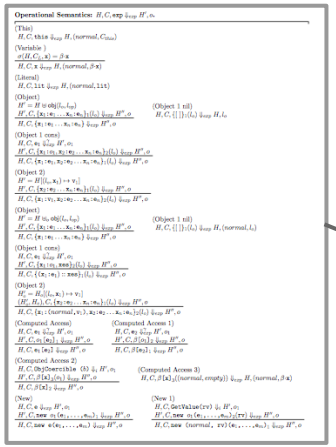
Today's Talk



- Difficulties with verification tool directly for JavaScript
- JS-IVL, an intermediate verification language for JavaScript
- Translation from JavaScript to JS-IVL
- Program Logic for JS-IVL

operational semantics of ES5Strict

Our Choices



JavaScript

EcmaScript 5 Strict (ES5Strict)

Principled Translation

JS-IVL

- Small language
- Simple semantics
- Similar memory model to JavaScript's

An Example

```
function Person (name) {
  this.name = name;
}

Person.prototype.sayHi = function () {
  return "Hi " + this.name
}

var alice = new Person ("Alice");
alice.sayHi()
```

An Example

```
function Person (name) {
  this.name = name;
}

Person.prototype.sayHi = function () {
  return "Hi " + this.name
}

var alice = new Person ("Alice");
alice.sayHi()
```

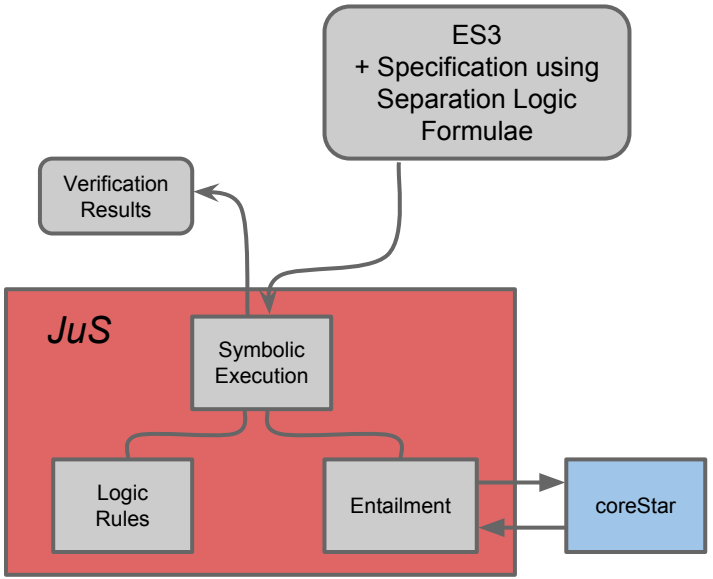
An Example

```
function Person (name) {
  this.name = name;
}

Person.prototype.sayHi = function () {
  return "Hi " + this.name
}

var alice = new Person ("Alice");
alice.sayHi()
```

An Example

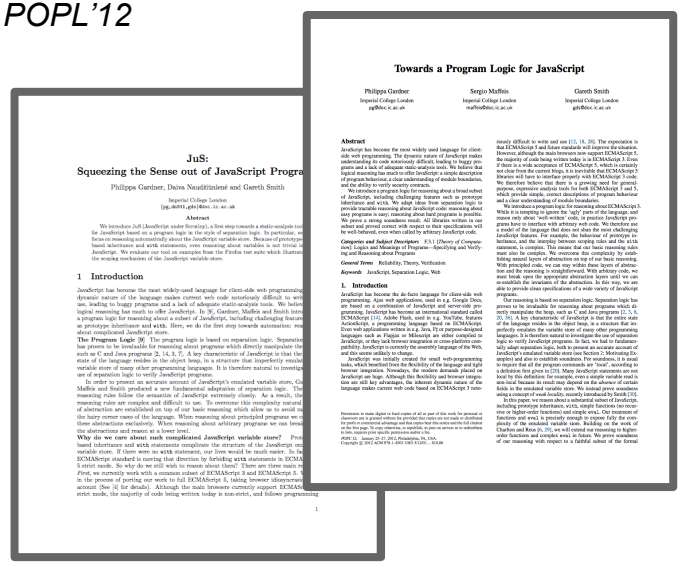
```
function Person (name) {
  this.name = name;
}

Person.prototype.sayHi = function () {
  return "Hi " + this.name
}

var alice = new Person ("Alice");
alice.sayHi()
```

The JuS Tool



JuS: Squeezing the Sense out of JavaScript Programs.
Gardner, Naudziuniene, Smith. ***JSTools@ECOOP'13***

Towards a Program Logic for JavaScript.
Gardner, Maffeis, Smith. POPL'12

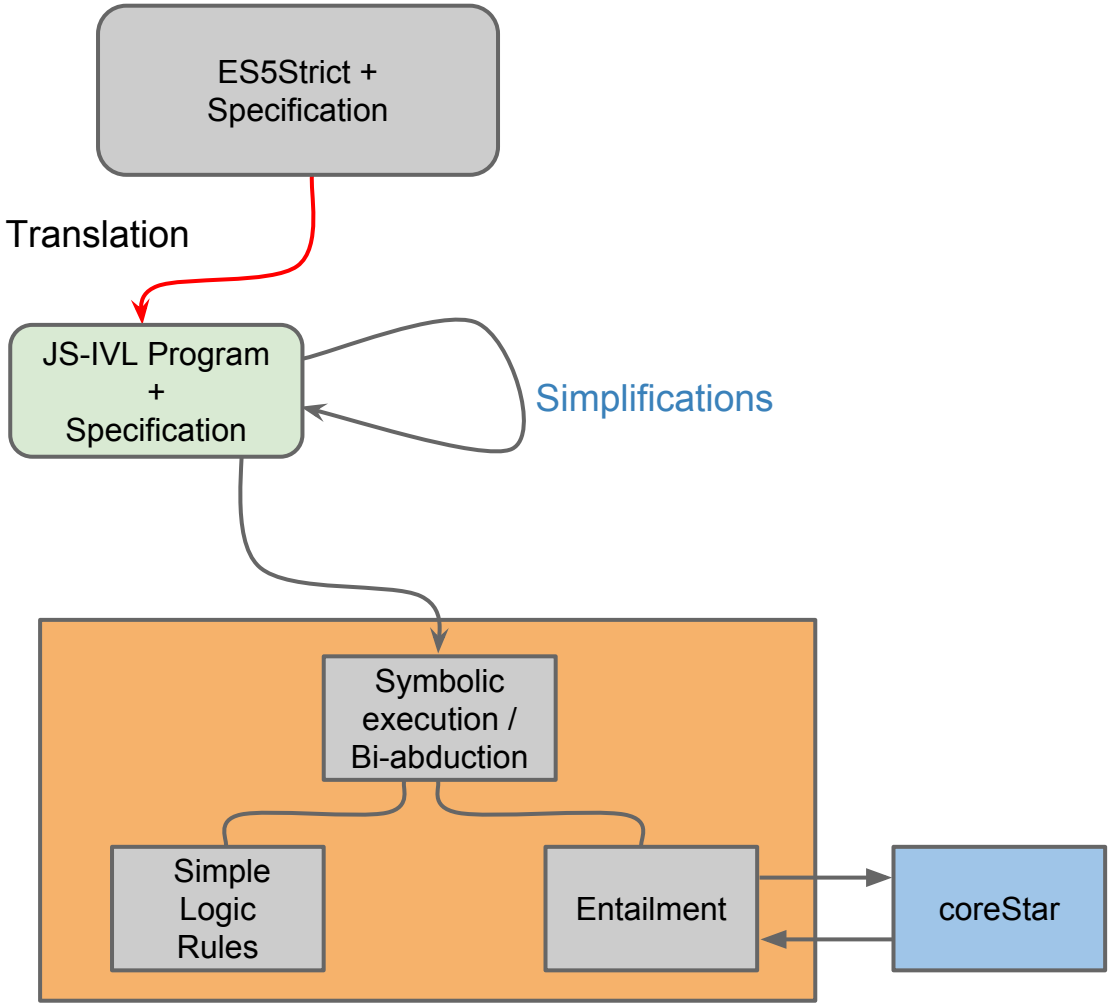


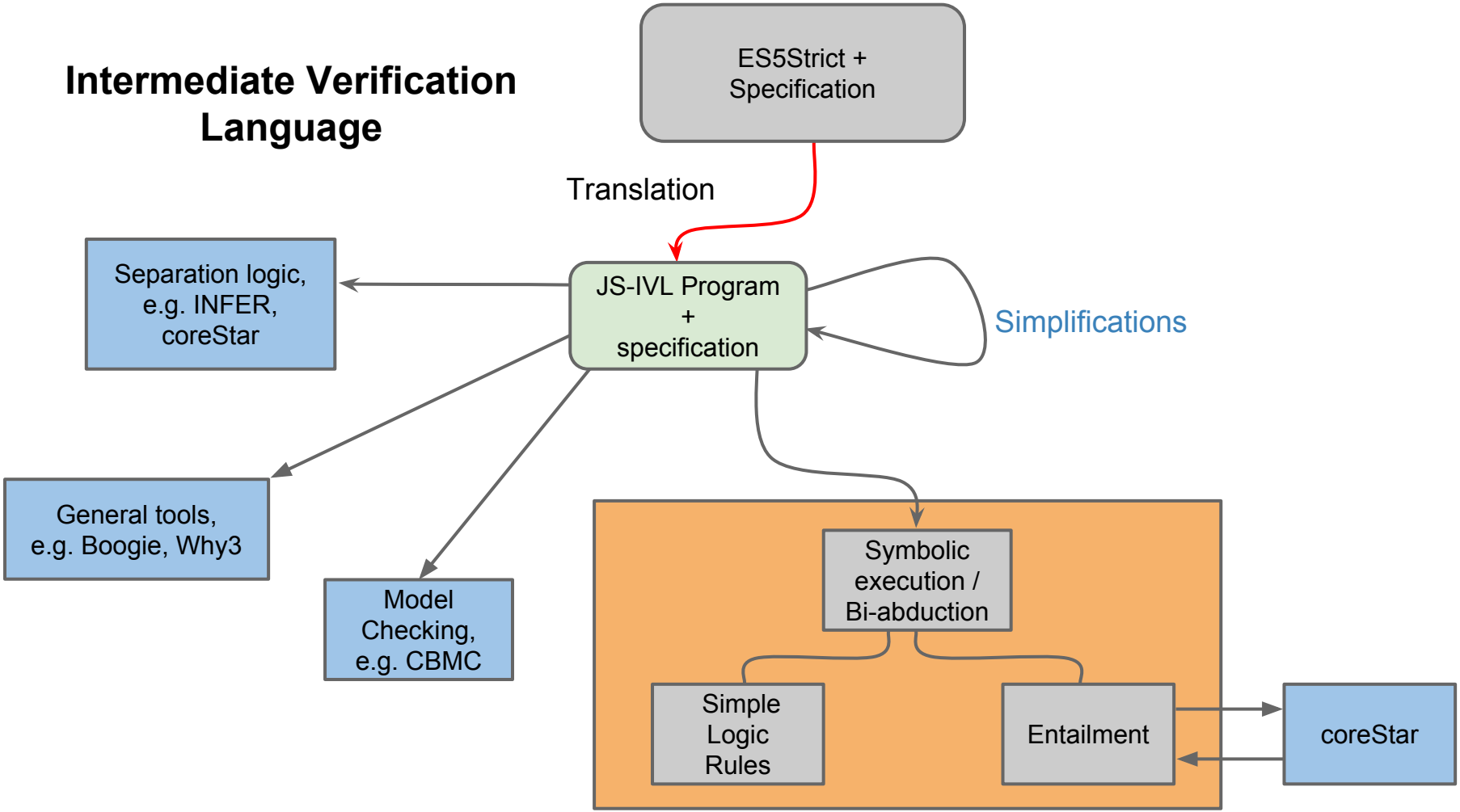
Takeaway from JuS

- JuS executes JavaScript programs symbolically and checks if given specification is correct
  - However, VERY complex Logic Rules do not scale
    - to implement **Bi-Abduction** to reduce annotations size
    - to support bigger and fuller subset of JavaScript

$$
\begin{array}{l}
(1)\mathit{Asm} \vdash \{P\}\,\mathtt{e}\,\{R_0 * \mathtt{r} \doteq F_1\} \\
(2)\ R_0 = \left(\begin{array}{l} S_0 \bowtie \mathsf{This}(F_1, T) \bowtie \gamma(Ls, F_1, F_2) * F_2 \neq l_e * \\ (F_2, @body) \mapsto \lambda X_1, \ldots, X_n.\mathtt{e}' * (F_2, @scope) \mapsto Ls_V' \end{array}\right) \\
(3.1)\ \mathit{Asm} \vdash \{R_0\}\,\mathtt{e1}\,\{R_1 * \boldsymbol{l} \doteq Ls_V * \mathtt{r} \doteq V_1\} \quad R_1 = S_1 * \gamma(Ls_1, V_1, V_1') \\
\vdots \\
(3.m)\ \mathit{Asm} \vdash \{R_{m-1}\}\,\mathtt{em}\,\{R_m * \boldsymbol{l} \doteq Ls_V * \mathtt{r} \doteq V_m\} \quad R_m = S_m * \gamma(Ls_m, V_m, V_m') \\
(4)\ \forall j \in \{m+1 \ldots n\}\ .\ V_j' = \&\mathtt{undefined} \\
(5)\ R_m' = \left(\begin{array}{l} R_m * \exists \mathrm{L}\ .\ \boldsymbol{l} \doteq \mathrm{L}{:}Ls_V' * \\ (\mathrm{L}, X_1) \mapsto V_1' * \\ \vdots \\ (\mathrm{L}, X_n) \mapsto V_n' * \\ (\mathrm{L}, @this) \mapsto T * \\ (\mathrm{L}, @proto) \mapsto \mathtt{null} * \mathsf{defs}(\mathrm{L}, \mathtt{e}', [X_1, \ldots, X_n]) * \\ \mathsf{decls}(\mathtt{e}', YS) * \mathsf{newobj}_{\mathrm{L}}(\{@proto, @this, X_1, \ldots, X_n\} \cup YS) \end{array}\right) \\
(6)\ \lambda \mathrm{X}_1 \ldots \mathrm{X}_n\ .\ \{\mathrm{P}_f\}\,\mathtt{e}'\,\{\mathrm{Q}_f\} \in \mathit{Asm} \qquad (7)\ \boldsymbol{l} \notin \mathsf{fv}(Q) \cup \mathsf{fv}(R_m) \\
\underline{(8)\ (\lambda \mathrm{X}_1 \ldots \mathrm{X}_n\ .\ \{\mathrm{P}_f\}\,\mathtt{e}'\,\{\mathrm{Q}_f\})\ V_1' \ldots V_n' \quad = \quad \{R_m'\}\,\mathtt{e}'\,\{\exists \mathrm{L}\ .\ Q * \boldsymbol{l} \doteq \mathrm{L}{:}Ls_V'\}} \\
\mathit{Asm} \vdash \{P\}\,\mathtt{e(e1,\ldots em)}\,\{\exists \mathrm{L}\ .\ Q * \boldsymbol{l} \doteq Ls_V\}
\end{array}
$$

Intermediate Verification Language




Intermediate Verification Language
ES5Strict + Specification
Translation
Separation logic, e.g. INFER, coreStar
JS-IVL Program + specification
Simplifications
General tools, e.g. Boogie, Why3
Model Checking, e.g. CBMC
Symbolic execution / Bi-abduction
Simple Logic Rules
Entailment
coreStar

JS-IVL

$$C \in \mathtt{Cmd} \triangleq \mathtt{r := E} \mid \mathtt{skip} \mid \mathtt{label\ l} \mid \mathtt{goto\ l} \mid \mathtt{goto\ [E]\ } l_1, l_2$$

Simple goto language, where l denotes a label, E is side effect free expression

JS-IVL

```
C ∈ Cmd ≜ r := E
  | skip | label l | goto l | goto [E] l1, l2
  | r := p(E,...,E) with l
```

Procedure calls, where p ∈ { E, eval, built-in-fid }

JS-IVL

```
C ∈ Cmd ≜ r := E
  | skip | label l | goto l | goto [E] l1, l2
  | r := p(E,...,E) with l
  | r := new() | r := hasField (E, E)
  | r := [E, E]| [E, E] := E
  | r := delete (E, E)
```

JavaScript Heap

JS-IVL

```
C ∈ Cmd ≜ r := E
  | skip | label l | goto l | goto [E] l1, l2
  | r := p(E,...,E) with l
  | r := new() | r := hasField (E, E)
  | r := [E, E]| [E, E] := E
  | r := delete (E, E)
  | r := protoField (E, E) | r := protoObj (E, E)
```

Prototype-based Inheritance

JS-IVL

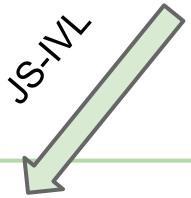
$C \in Cmd \triangleq r := E$
$\quad | \ skip \ | \ label \ l \ | \ goto \ l \ | \ goto \ [E] \ l_1, \ l_2$
$\quad | \ r := p(E,...,E) \ with \ l$
$\quad | \ r := new() \ | \ r := hasField \ (E, E)$
$\quad | \ r := [E, E] | \ [E, E] := E$
$\quad | \ r := delete \ (E, E)$
$\quad | \ r := protoField \ (E, E) \ | \ r := protoObj \ (E, E)$

$Procedure \triangleq procedure \ id \ (rthis, \ rscope, \ x_1,...,x_n)$
{
$\quad C_1;...;C_n$
}

JS-IVL Logic Rule for Procedure Call

$$\frac{\lambda x_1, \ldots, x_n.\{P\}\texttt{fid}\{Q * \texttt{r} \doteq V\} \in \mathcal{ASM} \qquad \forall j \in \{m+1 \ldots n\} \,.\, E_j = \texttt{\&undefined}}{\mathcal{ASM} \vdash \{P[E_1/x_1, \ldots, E_n/x_n]\}\, \texttt{x := fid(E}_1\texttt{, ...E}_m\texttt{) with lab}\, \{Q * \texttt{x} \doteq V[E_1/x_1, \ldots, E_n/x_n]\}}$$

$$
\begin{array}{l}
(1)\mathcal{Asm} \vdash \{P\}\ \mathtt{e}\ \{R_0 * \mathbf{r} \doteq F_1\} \\
(2)\ R_0 = \begin{pmatrix} S_0 \bowtie \mathsf{This}(F_1, T) \bowtie \gamma(Ls, F_1, F_2) * F_2 \not\doteq l_e * \\ (F_2, @body) \mapsto \lambda X_1, \ldots, X_n.\mathtt{e}' * (F_2, @scope) \mapsto Ls'_V \end{pmatrix} \\
(3.1)\ \mathcal{Asm} \vdash \{R_0\}\ \mathtt{e1}\ \{R_1 * \boldsymbol{l} \doteq Ls_V * \mathbf{r} \doteq V_1\} \quad R_1 = S_1 * \gamma(Ls_1, V_1, {V_1}') \\
\vdots \\
(3.m)\ \mathcal{Asm} \vdash \{R_{m-1}\}\ \mathtt{em}\ \{R_m * \boldsymbol{l} \doteq Ls_V * \mathbf{r} \doteq V_m\} \quad R_m = S_m * \gamma(Ls_m, V_m, {V_m}') \\
(4)\ \forall j \in \{m+1 \ldots n\}\ .\ {V_j}' = \mathtt{\&undefined} \\
(5)\ R'_m = \begin{pmatrix} R_m * \exists \mathrm{L}\ .\ \boldsymbol{l} \doteq \mathrm{L}{:}Ls'_V * \\ (\mathrm{L}, X_1) \mapsto {V_1}' * \\ \vdots \\ (\mathrm{L}, X_n) \mapsto {V_n}' * \\ (\mathrm{L}, @this) \mapsto T * \\ (\mathrm{L}, @proto) \mapsto \mathtt{null} * \mathsf{defs}(\mathrm{L}, \mathtt{e}', [X_1, \ldots, X_n]) * \\ \mathsf{decls}(\mathtt{e}', YS) * \mathsf{newobj}_{\mathrm{L}}(\{@proto, @this, X_1, \ldots, X_n\} \cup YS) \end{pmatrix} \\
(6)\ \lambda \mathrm{X}_1 \ldots \mathrm{X}_n\ .\ \{\mathrm{P}_f\}\ \mathtt{e}'\ \{\mathrm{Q}_f\} \in \mathcal{Asm} \qquad (7)\ \boldsymbol{l} \notin \mathsf{fv}(Q) \cup \mathsf{fv}(R_m) \\
(8)\ (\lambda \mathrm{X}_1 \ldots \mathrm{X}_n\ .\ \{\mathrm{P}_f\}\ \mathtt{e}'\ \{\mathrm{Q}_f\})\ V'_1 \ldots V'_n \quad = \quad \{R'_m\}\ \mathtt{e}'\ \{\exists \mathrm{L}\ .\ Q * \boldsymbol{l} \doteq \mathrm{L}{:}Ls'_V\} \\
\hline
\mathcal{Asm} \vdash \{P\}\ \mathtt{e(e1, \ldots em)}\ \{\exists \mathrm{L}\ .\ Q * \boldsymbol{l} \doteq Ls_V\}
\end{array}
$$

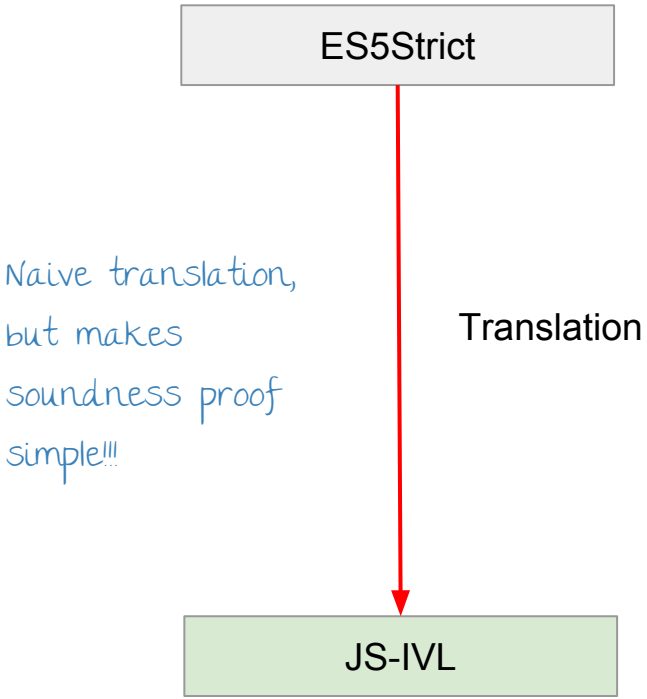


Logic Rules of Function / Procedure Call

$$
\begin{array}{l}
\lambda x_1, \ldots, x_n.\{P\}\mathtt{fid}\{Q * \mathbf{r} \doteq V\} \in \mathcal{ASM} \\
\forall j \in \{m+1 \ldots n\}\ .\ E_j = \mathtt{\&undefined} \\
\hline
\mathcal{ASM} \vdash \{P[E_1/x_1, \ldots, E_n/x_n]\}\ \mathtt{x\ :=\ fid(E_1,\ \ldots E_m)\ with\ lab}\ \{Q * \mathbf{x} \doteq V[E_1/x_1, \ldots, E_n/x_n]\}
\end{array}
$$

Complexity of the language

does not disappear, but has moved to the translation



Naive translation,
but makes
soundness proof
simple!!!

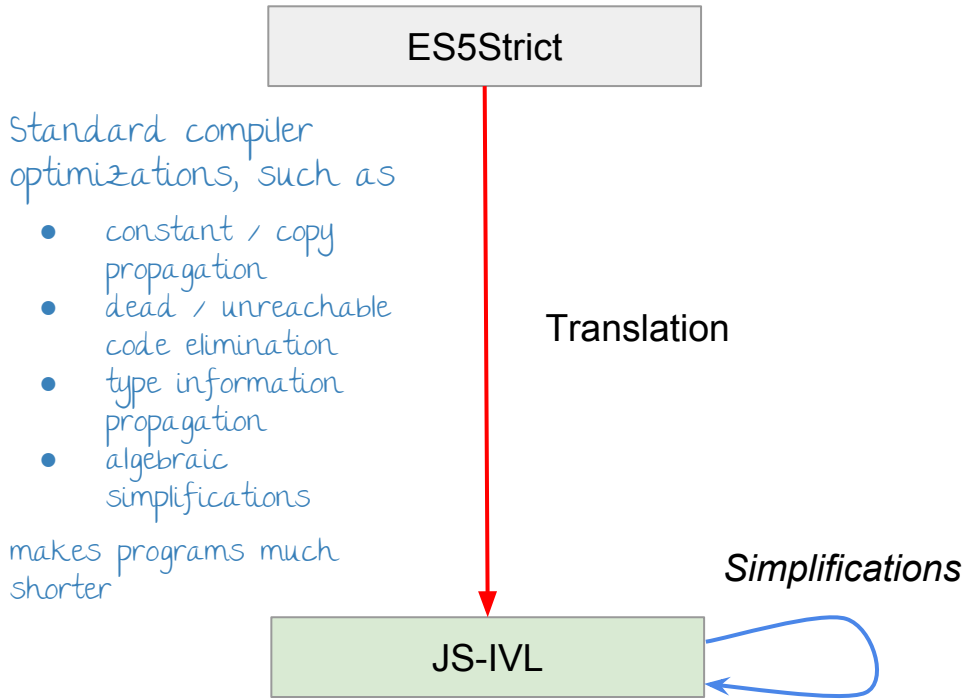
```
function Person (name) {
  this.name = name;
}
Person.prototype.sayHi = function () {
  return "Hi " + this.name
}
var alice = new Person ("Alice");
alice.sayHi()
```

```
procedure anonymous0 (rthis,rscope) {
 0. main_scope := [rscope,"main"]
1. anonymous0_scope := new ()
2. [anonymous0_scope,"#proto"] := null
3. r507 := #empty
4. r509 := rthis
5. goto [typeOf (r509) = Reference]
r580, r581
6. label r580
7. r511 := base (r509)
8. goto [r511 = #undefined] r583, r584
9. label r583
10. r525 := new ()
11. [r525,"#proto"] := #lrep
12. [r525,"#class"] := "Error"
13. r505 := r525
14. goto throw.r503
15. goto r585
16. label r584
17. r512 := field (r509
18. goto [typeOf (r511
typeOf (r511) = Number or typeOf
(r511) = String] r586, r587
19. label r586
```

...

```
1131. goto r463
1132. label r462
1133. r144 := r138
1134. goto r463
1135. label r463
1136. goto [r144 = #empty] r497, r498
1137. label r497
1138. r160 := r96
1139. goto r499
1140. label r498
1141. r160 := r144
1142. goto r499
1143. label r499
1144. goto [r160 = #empty] r500, r501
1145. label r500
1146. r3 := #undefined
1147. goto r502
1152. goto return.r1
1153. label return.r1
1154. label throw.r0
}
```

3183 lines



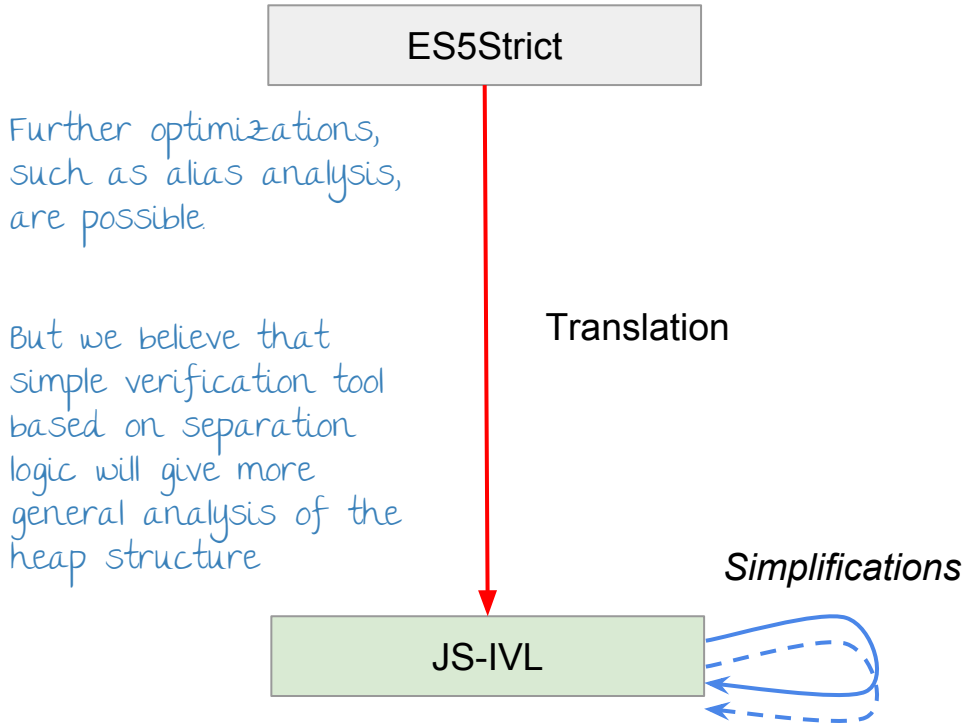

```
function Person (name) {
  this.name = name;
}
Person.prototype.sayHi = function () {
  return "Hi " + this.name
}
var alice = new Person ("Alice");
alice.sayHi()
```

```
procedure anonymous0 (rthis,rscope) {
 0. main_scope := [rscope,"main"]
1. anonymous0_scope := new ()
2. [anonymous0_scope,"#proto"] := null
3. r507 := #empty
4. r509 := rthis
5. goto [typeOf (r509) = Reference]
r580, r581
6. label r580
7. r511 := base (r509)
8. goto [r511 = #undefined] r583, r584
9. label r583
10. r525 := new ()
11. [r525,"#proto"] := #lrep
12. [r525,"#class"] := "Error"
13. r505 := r525
14. goto throw.r503
15. goto r585
16. label r584
17. r512 := field (r509
18. goto [typeOf (r511
typeOf (r511) = Number or typeOf
(r511) = String] r586, r587
19. label r586
```

...

```
1131. goto r463
1132. label r462
1133. r144 := r138
1134. goto r463
1135. label r463
1136. goto [r144 = #empty] r497, r498
1137. label r497
1138. r160 := r96
1139. goto r499
1140. label r498
1141. r160 := r144
1142. goto r499
1143. label r499
1144. goto [r160 = #empty] r500, r501
1145. label r500
1146. r3 := #undefined
1147. goto r502
1153. label return.r1
1154. label throw.r0
}
```

944 lines




```
function Person (name) {
  this.name = name;
}
Person.prototype.sayHi = function () {
  return "Hi " + this.name
}
var alice = new Person ("Alice");
alice.sayHi()
```

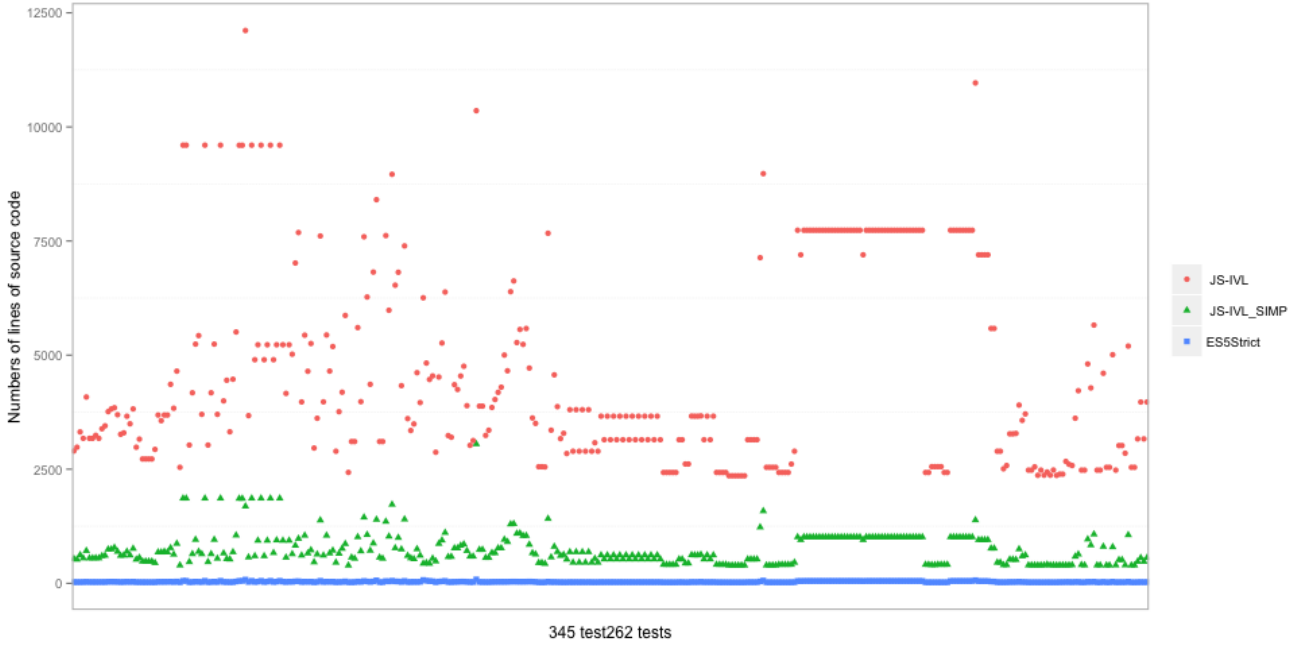
```
procedure anonymous0 (rthis,rscope) {
 0. main_scope := [rscope,"main"]
1. anonymous0_scope := new ()
2. [anonymous0_scope,"#proto"] := null
3. r507 := #empty
4. r509 := rthis
5. goto [typeOf (r509) = Reference]
r580, r581
6. label r580
7. r511 := base (r509)
8. goto [r511 = #undefined] r583, r584
9. label r583
10. r525 := new ()
11. [r525,"#proto"] := #lrep
12. [r525,"#class"] := "Error"
13. r505 := r525
14. goto throw.r503
15. goto r585
16. label r584
17. r512 := field (r509
18. goto [typeOf (r511
typeOf (r511) = Number or typeOf
(r511) = String] r586, r587
19. label r586
```

...

```
1131. goto r463
1132. label r462
1133. r144 := r138
1134. goto r463
1135. label r463
1136. goto [r144 = #empty] r497, r498
1137. label r497
1138. r160 := r96
1139. goto r499
1140. label r498
1141. r160 := r144
1142. goto r499
1143. label r499
1144. goto [r160 = #empty] r500, r501
1145. label r500
1146. r3 := #undefined
1147. goto r502
1153. label return.r1
1154. label throw.r0
}
```

297 lines

Simplification Evaluation



We took 345 test262 tests to evaluate simplifications.

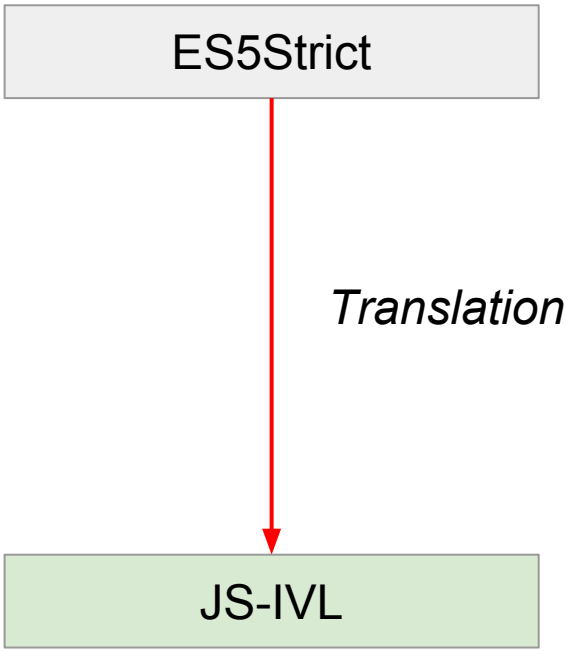
Line numbers on average:

ES5Strict : 33

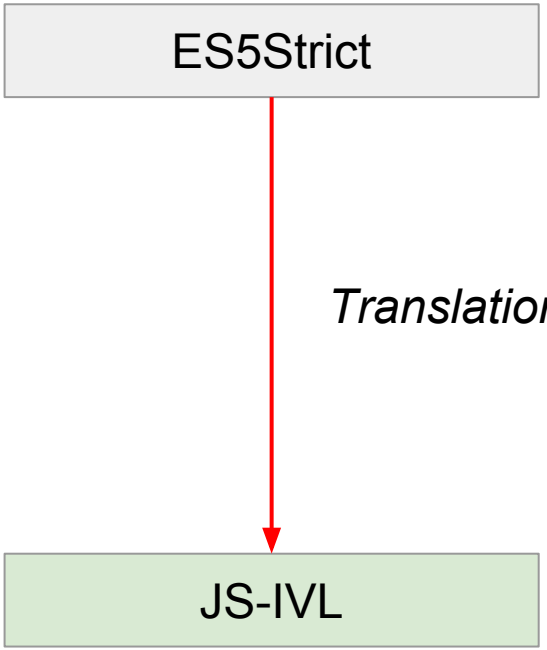
JS-IVL : 1601

JS-IVL simp: 738

Reliable Translation



Reliable Translation



1. Prove that translation is correct using operational semantics
2. Test using test262

Correctness Proof

Theorem 1 (Correctness)**.** Given a scope record Ψ and a function id id, a JavaScript heap H, a JS-IVL heap H', a scope chain L, a JS-IVL store ρ, a JavaScript statement s, a JS-IVL program $\mathbb{p}$, and a partial function $\beta : \mathcal{L}oc \rightharpoonup \mathcal{L}oc$, such that:

1. $\Psi \vdash H \sim_\beta H', \mathbb{p}$ (Hyp. 1)
2. $\Psi, id \vdash H, L \sim_\beta H', \rho$ (Hyp. 2)
3. $\mathsf{desc}(\mathbb{p}(id)), \Psi(id) \vdash s \searrow \Delta, \mathbf{x}$ (Hyp. 3)

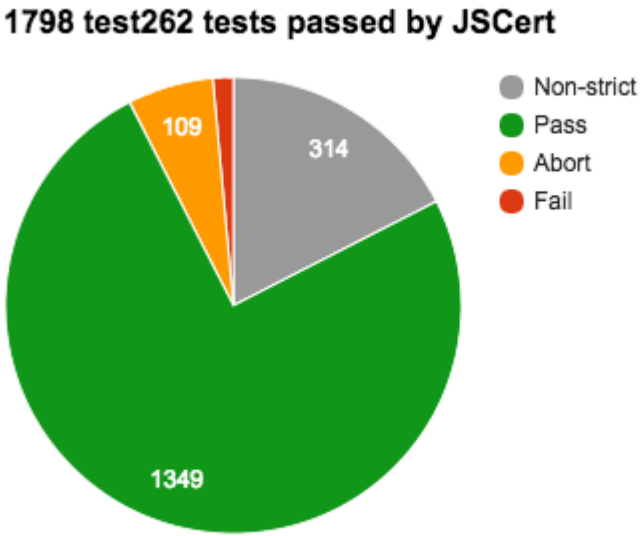
Then, it follows that:

$$\exists_{\langle H_f, v\rangle}\, L \vdash \langle H, s\rangle \Downarrow_e \langle H_f, v\rangle \quad \text{if and only if} \quad \exists_{\langle H'_f, \rho', v'\rangle^\ddagger}\, \mathbb{p}, \mathsf{desc}(\mathbb{p}(id)), \Delta \vdash \langle H', \rho, 1\rangle \Downarrow \langle H'_f, \rho', v'\rangle^\ddagger$$

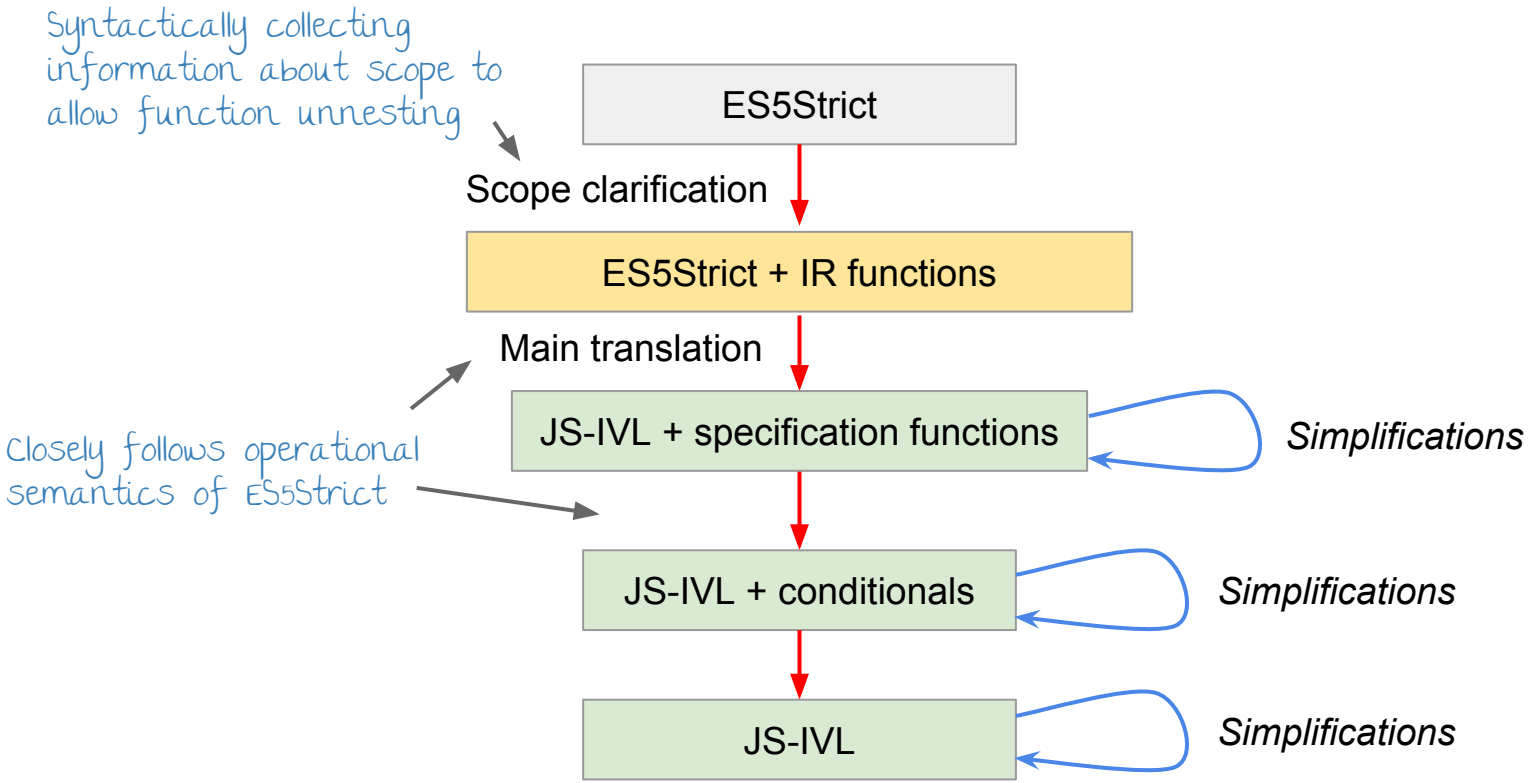
In which case, there is a function $\beta' : \mathcal{L}oc \rightharpoonup \mathcal{L}oc$ that extends β such that:

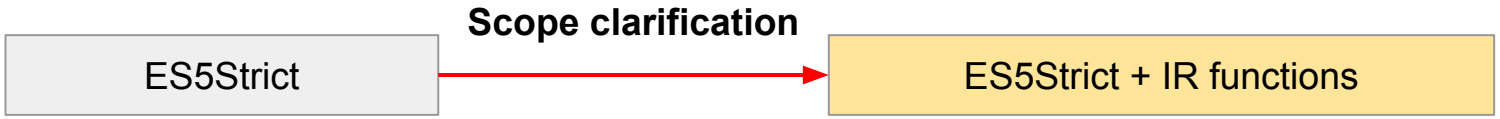
1. $\Psi \vdash H_f \sim_{\beta'} H'_f, \mathbb{p}$
2. $\Psi, id \vdash H_f, L \sim_{\beta'} H'_f, \rho'$
3. $v \sim_{\beta'} v' = \rho'(\mathbf{x})$

Testing Results



Translation






```
function Person (name) {
  this.name = name;
}
Person.prototype.sayHi = function () {
  return "Hi " + this.name
}
var alice = new Person ("Alice");
alice.sayHi()
```

```
IRfunction main () {
  function Person (name) { /** @fid "Person0" **/ }
  Person.prototype.sayHi = function () { /** @fid "anonymous1" **/ }
  var alice = new Person ("Alice");
  alice.sayHi()
} []

IRfunction Person0 (name) {
  this.name = name;
} [main: [Person, alice]]

IRfunction anonymous1 () {
  return "Hi " + this.name
} [main: [Person, alice]]
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    obj := new ()
    [obj,#proto] := person_proto
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (obj, scope, "Alice") with exl
    …
}
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
  …
  person := proto_field (#GlobalObject, "Person")
  …
  person_proto := [person, "prototype"]
  …
  obj := new ()
  [obj,#proto] := person_proto
  …
  constructor := [person, #fid]
  …
  scope := [person, #scope]
  …
  f_result := constructor (obj, scope, "Alice") with exl
  …
}
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    obj := new ()
    [obj,#proto] := person_proto
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (obj, scope, "Alice") with exl
    …
}
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
   …
   person := proto_field (#GlobalObject, "Person")
   …
   person_proto := [person, "prototype"]
   …
   obj := new ()
   [obj,#proto] := person_proto
   …
   constructor := [person, #fid]
   …
   scope := [person, #scope]

   f_result := constructor (obj, scope, "Alice") with exl
   …
}
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    obj := new ()
    [obj,#proto] := person_proto
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (obj, scope, "Alice") with exl
}
```

Main Translation

ES5Strict + IR functions → JS-IVL

```
IRfunction main () {
  function Person (name) {
    /** @fid "Person0" **/
  }
  Person.prototype.sayHi = function () {
    /** @fid "anonymous1" **/
  }
  var alice = new Person ("Alice");
  alice.sayHi()
} []
```

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    obj := new ()
    [obj,#proto] := person_proto
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (obj, scope, "Alice") with exl
    …
}
```

#undefined

Separation Logic

Frame rule

$$\frac{\{P\}\ c\ \{Q\}}{\{P * R\}\ c\ \{Q * R\}}$$

$$\mathbf{emp}$$

$$(X,Y) \mapsto Z$$
$$(X,Y) \mapsto \varnothing$$

$$H * H'$$

Some of JS-IVL Program Logic Rules

(Allocation)

$$\Gamma \vdash \{\oslash\}\ \mathtt{x\ :=\ new()}\ \{\exists L.\mathtt{x} \doteq L * \mathit{EmptyFields}(L, \{\})\}$$

(HasField True)

$$\Gamma \vdash \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto V * V \neq \varnothing\}\ \mathtt{x\ :=\ hasField(E_1, E_2)}\ \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto V * V \neq \varnothing * \mathtt{x} \doteq \mathtt{true}\}$$

(HasField False)

$$\Gamma \vdash \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto \varnothing\}\ \mathtt{x\ :=\ hasField(E_1, E_2)}\ \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto \varnothing * \mathtt{x} \doteq \mathtt{false}\}$$

(Lookup)

$$\Gamma \vdash \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto V * V \neq \varnothing\}\ \mathtt{x\ :=\ [E_1, E_2]}\ \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto V * V \neq \varnothing * \mathtt{x} \doteq V\}$$

(Update)

$$\Gamma \vdash \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto \_\}\ \mathtt{[E_1, E_2]\ :=\ E_3}\ \{(\mathtt{E}_1, \mathtt{E}_2) \mapsto \mathtt{E}_3\}$$

Symbolic execution

```
function Person (name) {
  this.name = name;
}
```

Symbolic execution. Give specification.

```
{ (this,"name") ↦ _ }
function Person (name) {
  this.name = name;
}
{ (this,"name") ↦ name * r ≐ undefined }
{ false }
```

Symbolic execution. Translating code and specification

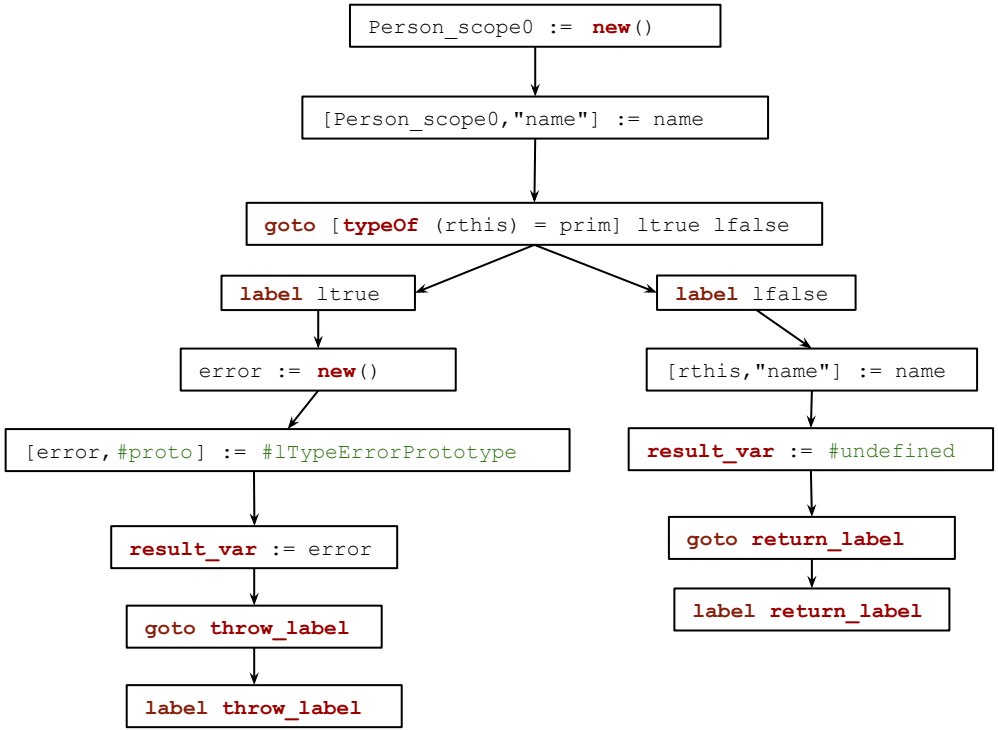
```
{ (this,"name") ↦ _ }
function Person (name) {
  this.name = name;
}
{ (this,"name") ↦ name * r ≐ undefined }
{ false }
```



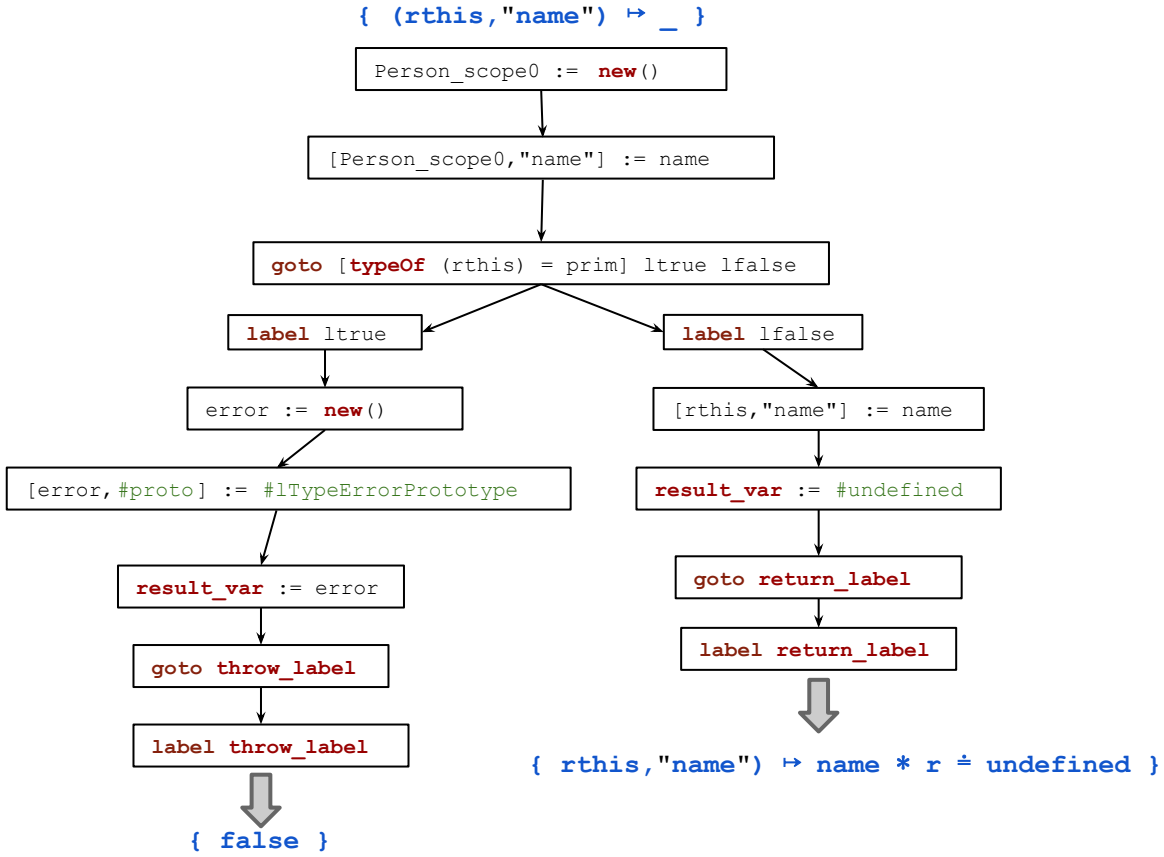
```
{ (rthis,"name") ↦ _ }

procedure Person0 (rthis, rscope, name) {
  Person_scope0 := new()
  [Person_scope0,"name"] := name
  goto [typeof (rthis) = #prim] ltrue lfalse

  label ltrue
  error := new()
  [error,#proto] := #lTypeErrorPrototype
  result_var := error
  goto throw_label

  label lfalse
  [rthis,"name"] := name
  result_var := #undefined
  goto return_label
}

{ (rthis,"name") ↦ name * r ≐ undefined }
{ false }
```

Example. Constructing control flow graph

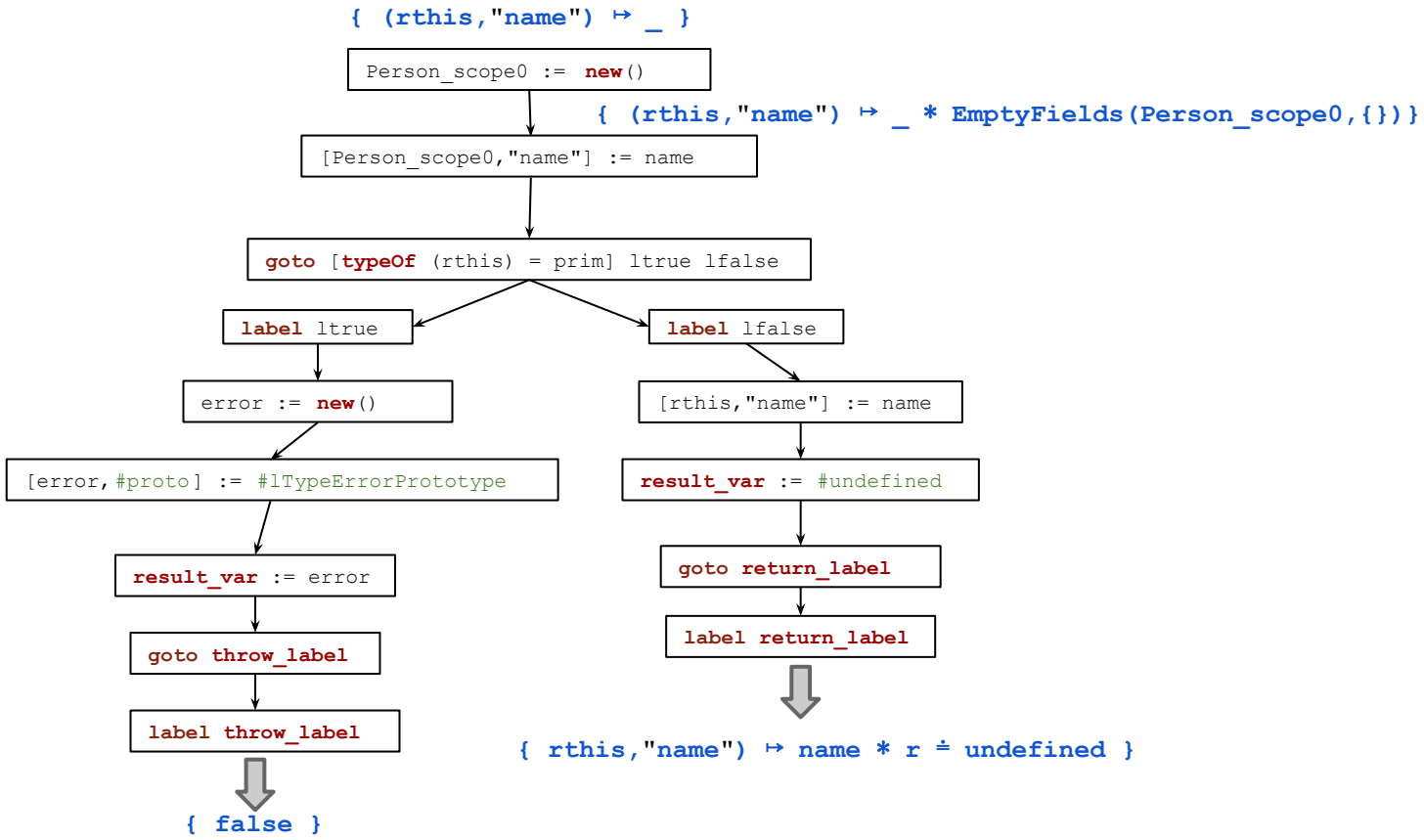
```
{ (rthis,"name") ↦ _ }

procedure Person0 (rthis, rscope, name) {
  Person_scope0 := new()
  [Person_scope0,"name"] := name
  goto [typeof (rthis) = #prim] ltrue lfalse

  label ltrue
  error := new()
  [error,#proto] := #lTypeErrorPrototype
  result_var := error
  goto throw_label

  label lfalse
  [rthis,"name"] := name
  result_var := #undefined
  goto return_label
}

{ (rthis,"name") ↦ name * r ≐ undefined }
{ false }
```



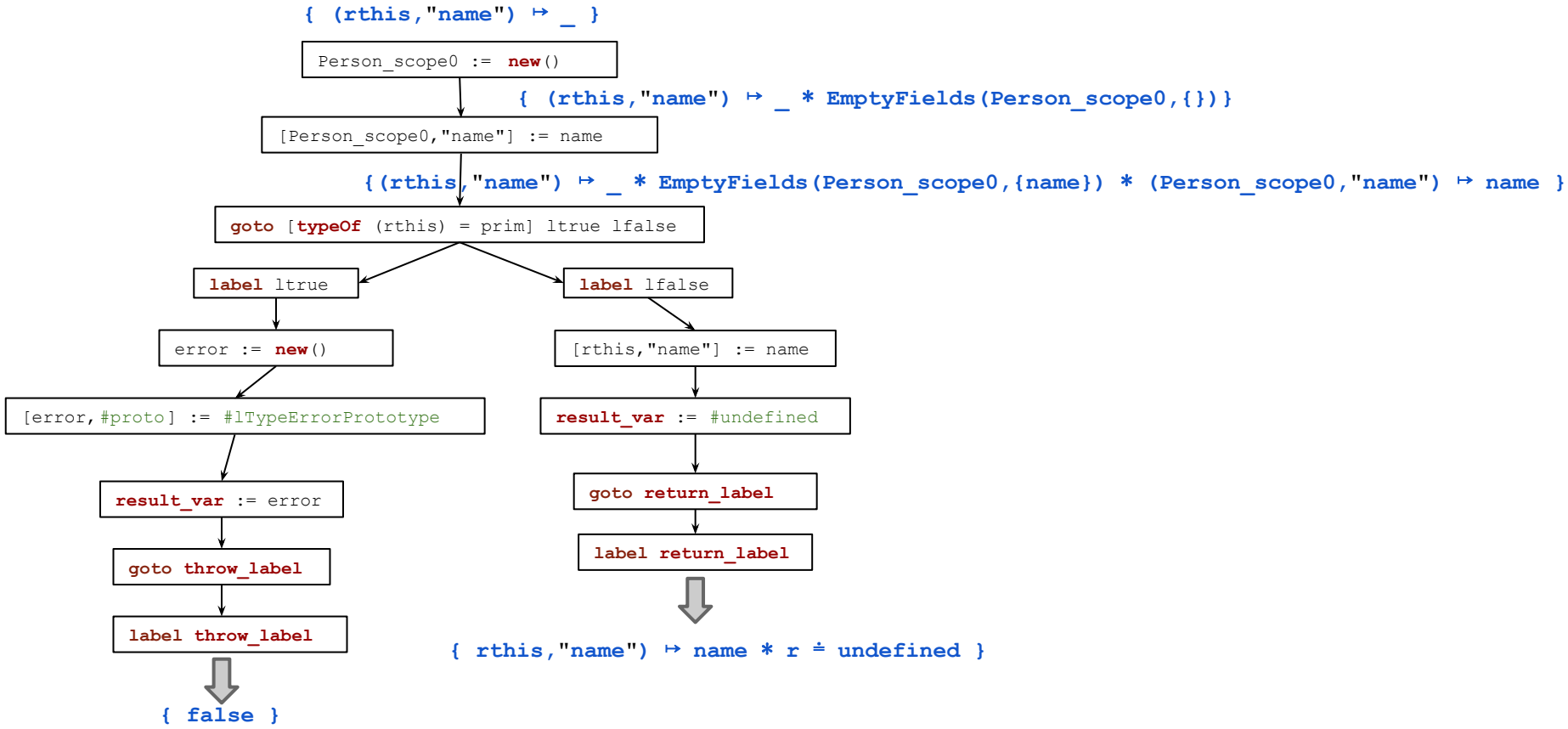
Example. Symbolic execution



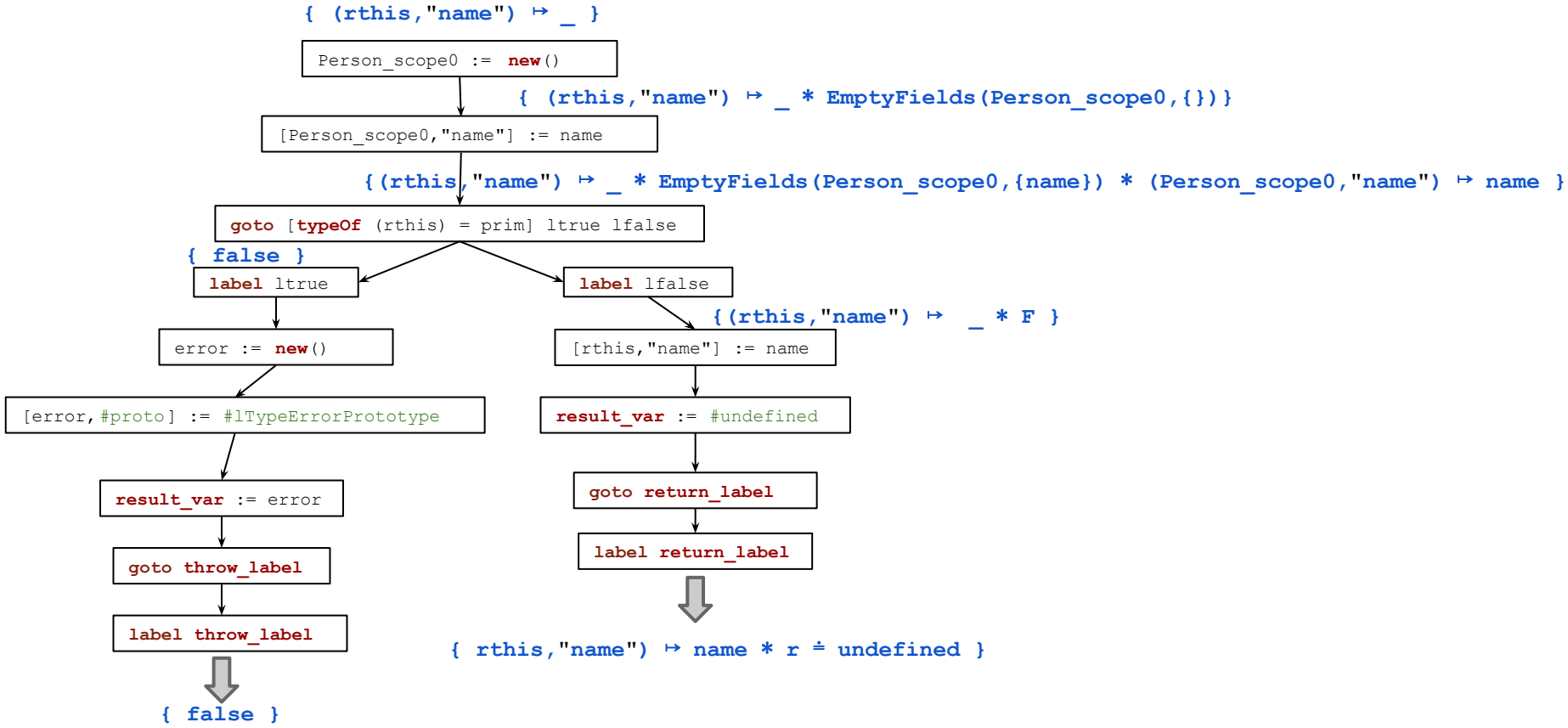
Example. Symbolic execution



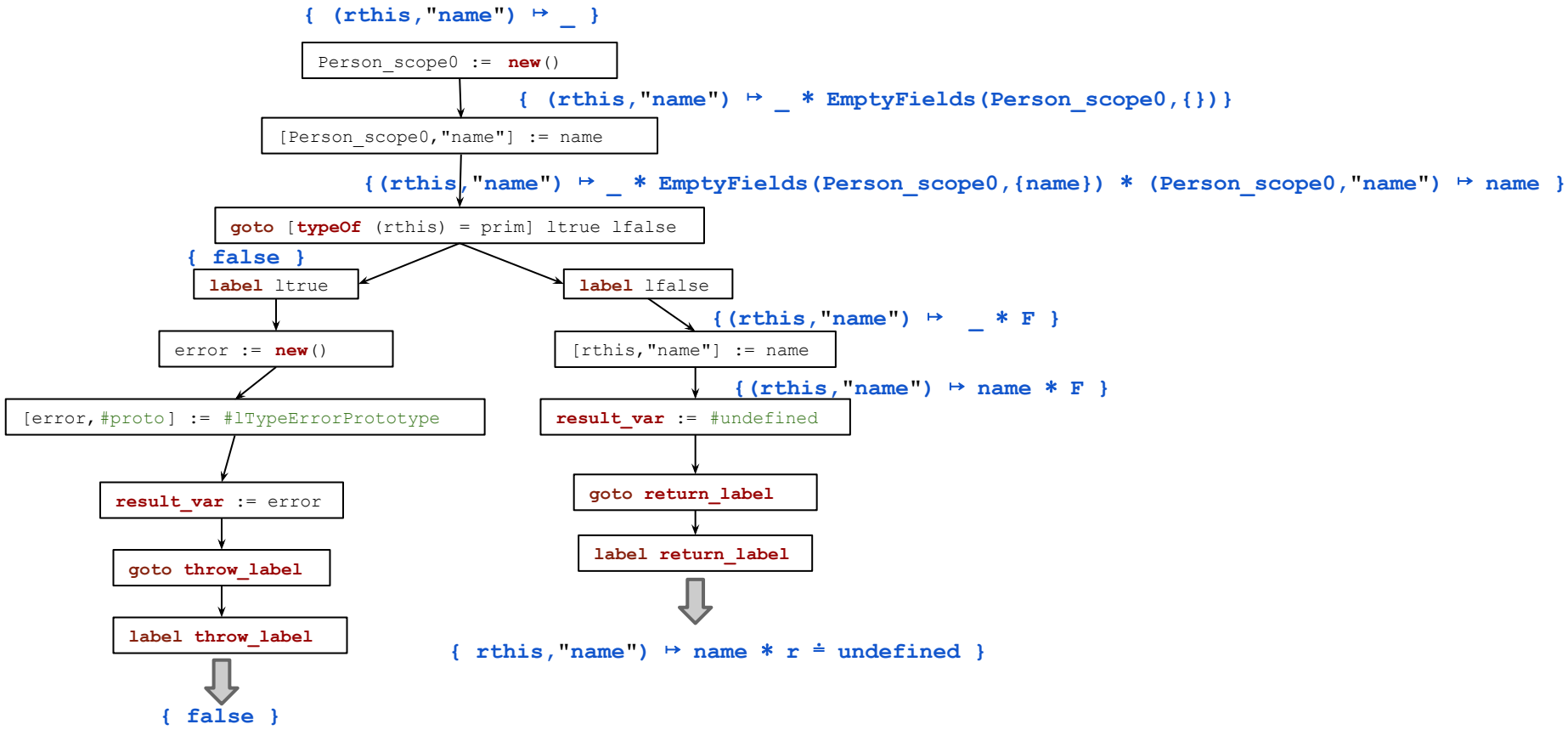
Example. Symbolic execution



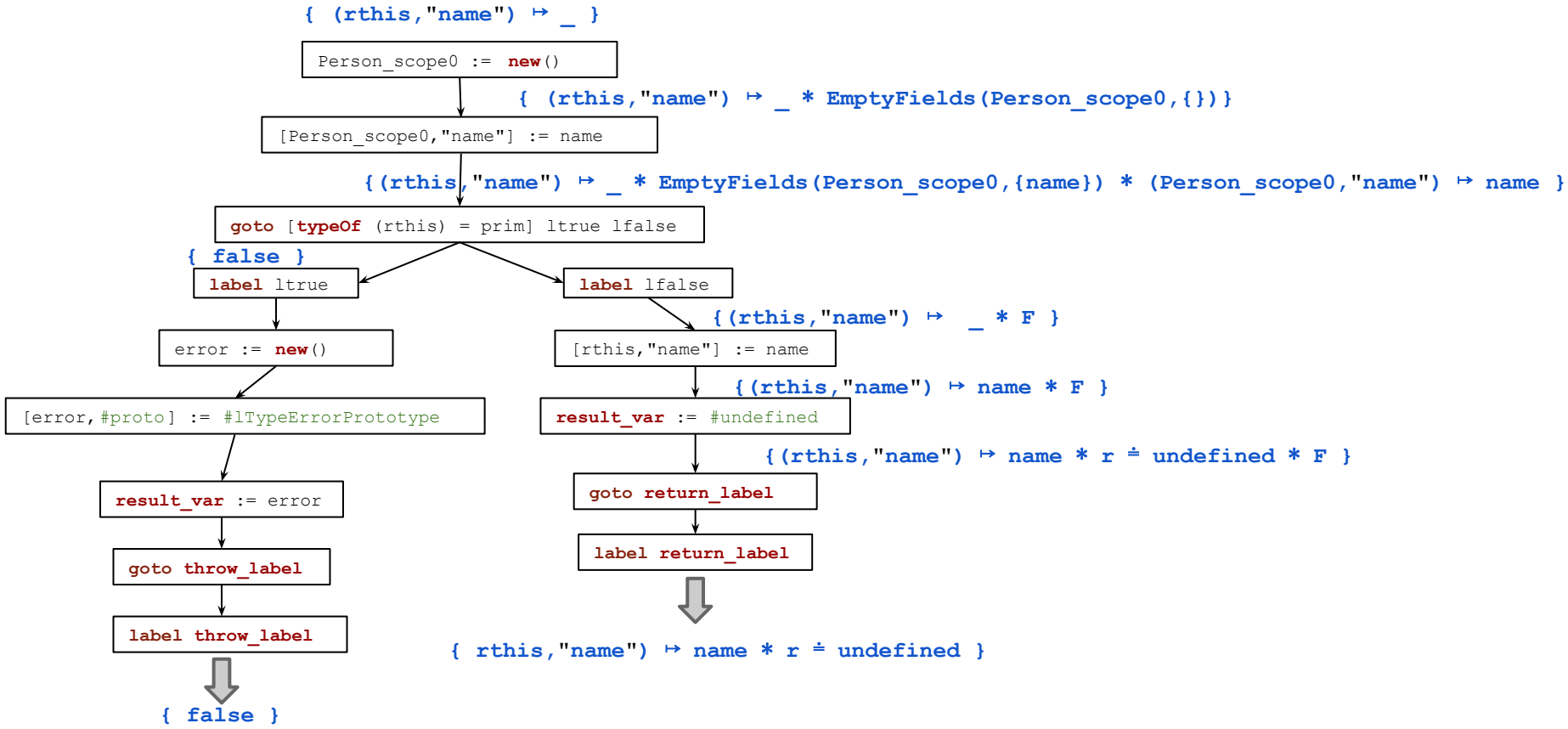
Example. Symbolic execution



Example. Symbolic execution



Example. Symbolic execution



Example. Symbolic execution

```
{ (rthis,"name")↦_ }
  Person0 (rthis, rscope, name)
{ (rthis,"name")↦name * r ≐ undefined }
```

Example. Symbolic execution

{ (rthis,"name")↦_ }

Person0 (rthis, rscope, name)

{ (rthis,"name")↦name * r ≐ undefined }

{ … * constructor ≐ Person0 * (obj,"name")↦ø }

{ … * (obj,"name")↦Alice * f_result ≐ undefined }

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    obj := new ()
    [obj,#proto] := person_proto
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (obj, scope, "Alice") with exl
    …
}
```

Example. Symbolic execution

{ (rthis,"name")↦_ }

Person0 (rthis, rscope, name)

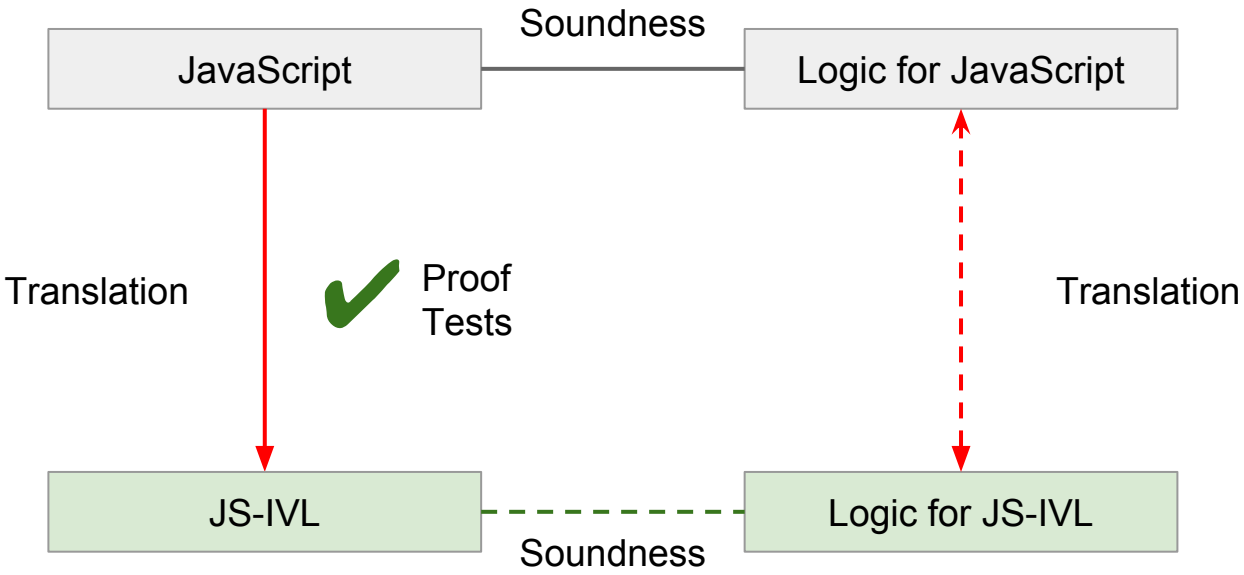
{ (rthis,"name")↦name * r ≐ undefined }

{ … * constructor ≐ Person0 }

Cannot apply spec

```
procedure main (rthis,rscope) {
    …
    person := proto_field (#GlobalObject, "Person")
    …
    person_proto := [person, "prototype"]
    …
    constructor := [person, #fid]
    …
    scope := [person, #scope]
    …
    f_result := constructor (#undefined, scope, "Alice") with exl
    …
}
```

Symbolic Execution



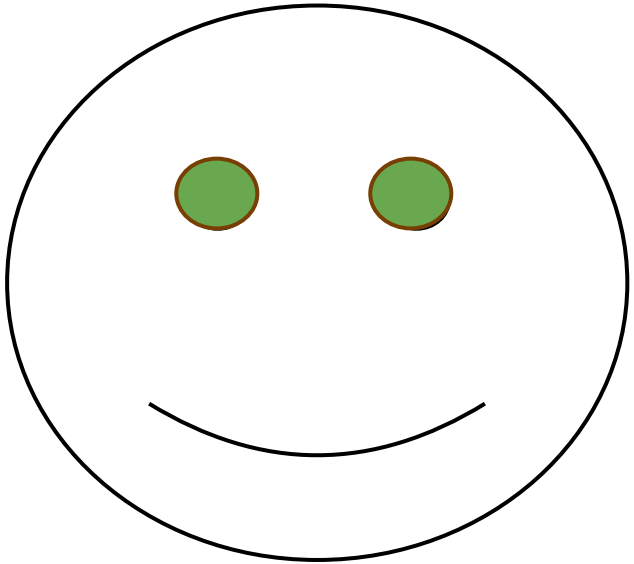
Conclusions

- JS-IVL, an intermediate verification language for ES5Strict
  - simple semantics
  - simple logic rules
- Translation to JS-IVL that follows ES5Strict operational semantics
- Correctness of the translation:
  - Soundness proof of the translation for a simplified setting
  - Testing implementation using test262 tests
- Program Logic for JS-IVL and symbolic execution tool (in progress)

Future work

- Implement bi-abduction
- Evaluate symbolic execution tool
- Experiment with different backends, e.g. separation logic (INFER, coreStar), general tools (Boogie, Why3), model checking (CBMC)

THE END



Thank you!

EcmaScript 5 Strict Syntax

We support all of ES5Strict, except of:

- Getters / Setters
- Switch statement
- Property attributes
- Arguments Object

Next to do

- For-in statement
- Indirect eval

Not doing

- Array literals and library
- Regular expression literals and library
- Other libraries (chapter 15)

use existing implementation, e.g, V8

$E ::= PE \mid \texttt{function}\ x_\perp(x,...,x)\{\ P\ \} \mid E[E] \mid E.x$

$\mid \texttt{new}\ E\ (E,...,E) \mid E\ (E,...,E) \mid E\texttt{++} \mid E\texttt{--} \mid U_{op}\ E$ — Expressions

$\mid E \oplus E \mid (E\ \texttt{?}\ E\ \texttt{:}\ E) \mid E \oplus_\perp \texttt{=}\ E \mid E\ \texttt{,}\ E$

$PE ::= \texttt{this} \mid x \mid Lit \mid \{PA,...,PA\} \mid \texttt{ArrayLiteral} \mid (E)$ — Primary Expressions

$Lit ::= \texttt{null} \mid \texttt{true} \mid \texttt{false} \mid \texttt{n} \mid \texttt{s} \mid \texttt{Regular Expression Literals}$ — Literals

$PA ::= PN : E \mid \texttt{get}\ PN\ (\ )\ \{\ P\ \} \mid \texttt{set}\ PN\ (\ x\ )\ \{\ P\ \}$ — Property Assignment

$PN ::= x \mid s \mid n$ — Property Name

$U_{op} ::= \texttt{delete} \mid \texttt{void} \mid \texttt{typeof} \mid \texttt{++} \mid \texttt{--} \mid \texttt{+} \mid \texttt{-} \mid \texttt{~} \mid \texttt{!}$ — Unary Operators

$\oplus ::= * \mid / \mid \% \mid + \mid - \mid \texttt{<<} \mid \texttt{>>} \mid \texttt{>>>} \mid < \mid > \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{instanceof}$

$\mid \texttt{in} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{===} \mid \texttt{!==} \mid \texttt{\&} \mid \texttt{\^{}} \mid \texttt{|} \mid \texttt{\&\&} \mid \texttt{||}$ — Binary Operators

$T ::= \{T\ ...\ T\} \mid \texttt{var}\ x\ \texttt{=}\ E_\perp,\ ...\ ,x\ \texttt{=}\ E_\perp \mid skip \mid E$

$\mid \texttt{if}\ (\ E\ )\ T\ \texttt{else}\ T_\perp \mid \texttt{do}\ T\ \texttt{while}\ (E) \mid \texttt{while}\ (E)\ T$

$\mid \texttt{for}\ (E_\perp;E_\perp;E_\perp)\ T$

$\mid \texttt{for}\ (\texttt{var}\ x\ \texttt{=}\ E_\perp,\ ...\ ,x\ \texttt{=}\ E_\perp;E_\perp;E_\perp)\ T$

$\mid \sout{\texttt{for}\ (\ E\ \texttt{in}\ E\ )\ T} \mid \sout{\texttt{for}\ (\texttt{var}\ x\ \texttt{=}\ E_\perp\ \texttt{in}\ E\ )\ T}$

$\mid \texttt{continue}\ x_\perp \mid \texttt{break}\ x_\perp \mid \texttt{return}\ E_\perp$

$\mid \texttt{switch}\ (E)\ \{$ — Statements

$\quad \texttt{case}_\perp\ E\ \texttt{:}\ \ T\ ...\ T$

$\quad \texttt{default}_\perp\ \texttt{:}\ \ T\ ...\ T$

$\quad \texttt{case}_\perp\ E\ \texttt{:}\ \ T\ ...\ T\ \}$

$\mid x\ \texttt{:}\ T \mid \texttt{throw}\ E \mid \texttt{try}\ \{T\ ...\ T\}\ \texttt{finally}\ \{T\ ...\ T\}$

$\mid \texttt{try}\ \{T\ ...\ T\}\ \texttt{catch}\ (x)\ \{T\ ...\ T\}\ \texttt{finally}_\perp\ \{T\ ...\ T\}$

$\mid \sout{\texttt{debugger ;}}$

$P ::= SE...SE$ — Program

$SE ::= T \mid \texttt{function}\ x(x,...,x)\{\ P\ \}$ — Source Elements