Certified Abstract Interpretation with Pretty-Big-Step Semantics

Martin Bodin   Thomas Jensen   Alan Schmitt

Inria

12–14th of January

CPP'15

Previously at POPL

JSCert: A Trusted Mechanised JavaScript Specification

jscert.org

- An operational semantics for JavaScript;
- Trusted;
- *Huge* ($\sim$ 800 reduction rules).

How to derive an abstract interpreter from such a huge semantics?

... proven in Coq?

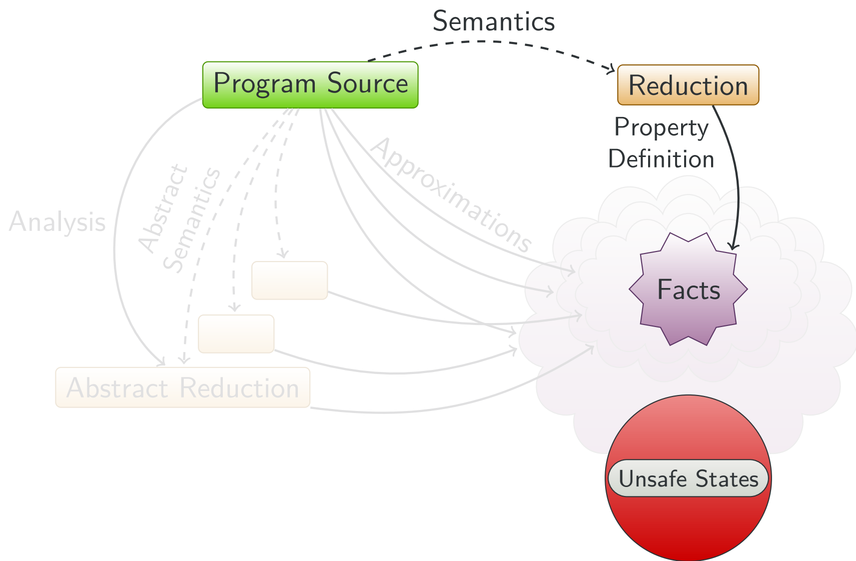
Let's make it correct *by construction*!

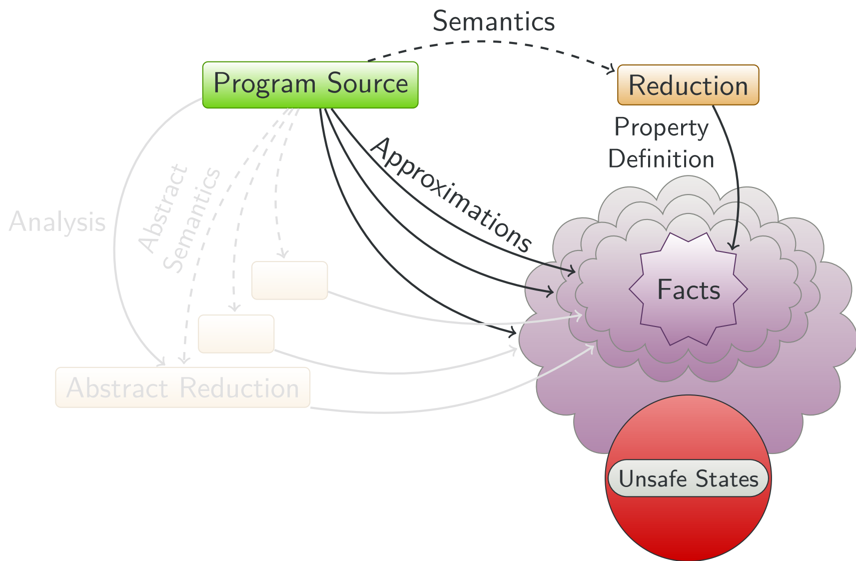
How to derive an abstract interpreter from such a huge semantics?

... proven in Coq?

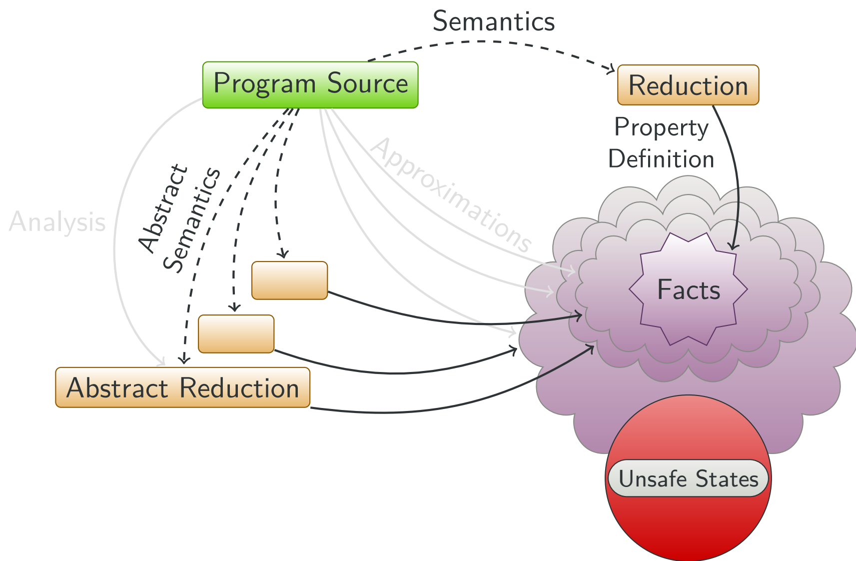
Let's make it correct *by construction*!

Abstract Interpretation

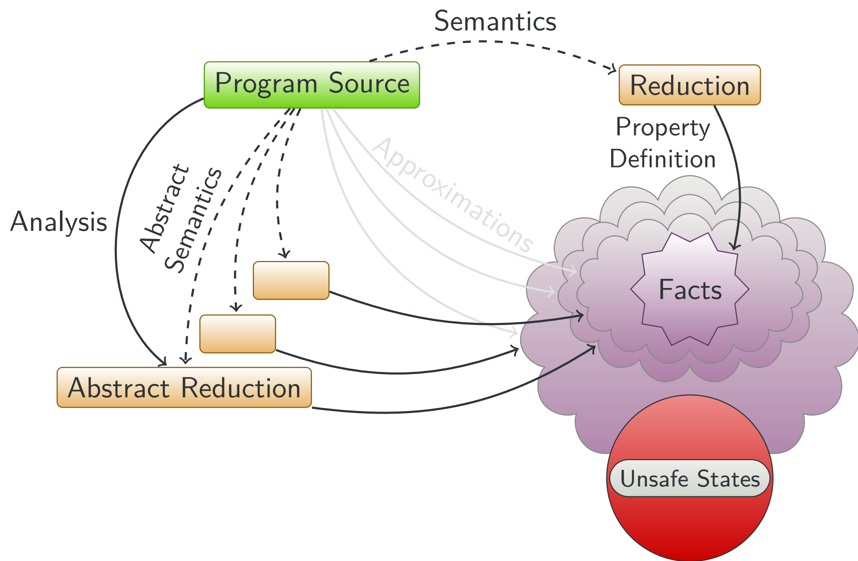



Semantics
Program Source
Reduction
Property
Definition
Approximations
Analysis
Abstract
Semantics
Facts
Abstract Reduction
Unsafe States

Abstract Interpretation



Abstract Interpretation



Concrete Domains

`int`, `bool`

Concrete Operations

`+`, `=`

Concrete Semantics

$t, \sigma \Downarrow r$

Abstract Domains

Sign

Abstract Operations

$+^\sharp$, $=^\sharp$

Abstract Semantics

$t, \sigma^\sharp \Downarrow^\sharp r^\sharp$

Abstract Interpreter

$f\left(t, \sigma^\sharp\right) = r^\sharp$

Concrete Domains

`int`, `bool`

Concrete Operations

`+`, `=`

Concrete Semantics

$t, \sigma \Downarrow r$

Abstract Domains

Sign

Abstract Operations

$+^\sharp$, $=^\sharp$

Abstract Semantics

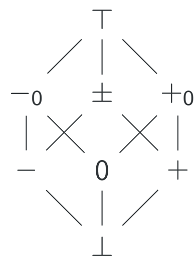
$t, \sigma^\sharp \Downarrow^\sharp r^\sharp$

Abstract Interpreter

$f\left(t, \sigma^\sharp\right) = r^\sharp$

Defining Abstract Domains

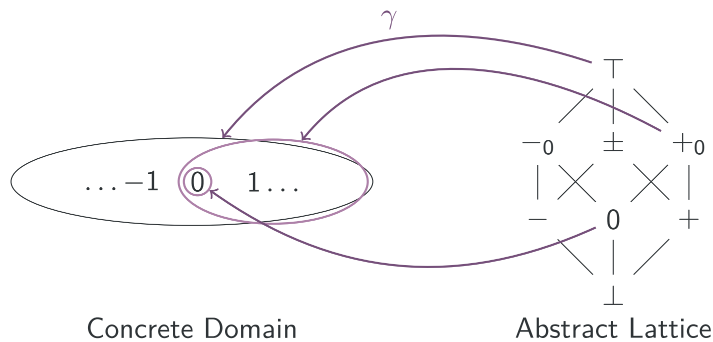
Concrete Domain



Abstract Lattice

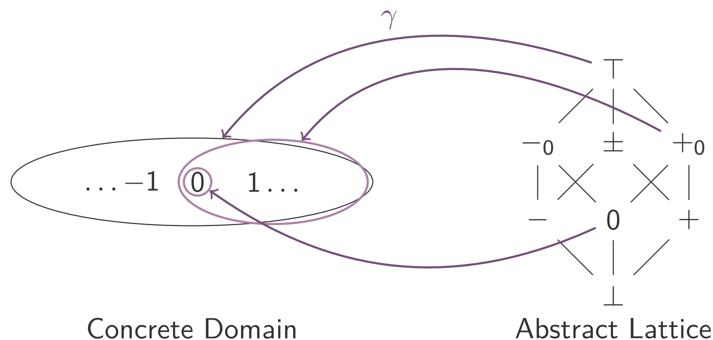
The theory has already been formalized in Coq.

Defining Abstract Domains



The theory has already been formalized in Coq.

Defining Abstract Domains



The theory has already been formalized in COQ.

Concrete Domains

`int`, `bool`

Concrete Operations

`+`, `=`

Concrete Semantics

$t, \sigma \Downarrow r$

Abstract Domains

Sign

Abstract Operations

$+^\sharp$, $=^\sharp$

Abstract Semantics

$t, \sigma^\sharp \Downarrow^\sharp r^\sharp$

Abstract Interpreter

$f\left(t, \sigma^\sharp\right) = r^\sharp$

Defining an Abstract Semantics, the Direct Approach

$$\textsc{IfTrue}\ \frac{e, E \Downarrow v \qquad s_1, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'}\ v \neq 0 \qquad \textsc{IfFalse}\ \frac{e, E \Downarrow v \qquad s_2, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'}\ v = 0$$

Let's just add $\sharp$ everywhere!

Defining an Abstract Semantics, the Direct Approach

$$\textsc{IfTrue}\ \frac{e, E \Downarrow v \qquad s_1, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'}\ v \neq 0 \qquad \textsc{IfFalse}\ \frac{e, E \Downarrow v \qquad s_2, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'}\ v = 0$$

Let's just add $\sharp$ everywhere!

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

$$\textsc{IfFalse}\ \frac{e, E^\sharp \Downarrow^\sharp v^\sharp \qquad s_2, E^\sharp \Downarrow^\sharp E'^\sharp}{if\, e\, s_1\, s_2, E^\sharp \Downarrow^\sharp E'^\sharp}\ \gamma\left(v^\sharp\right) \cap \{0\} \neq \emptyset$$

Defining an Abstract Semantics, the Direct Approach

$$\textsc{IfTrue}\quad \frac{e, E \Downarrow v \qquad s_1, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'} \quad v \neq 0$$

$$\textsc{IfFalse}\quad \frac{e, E \Downarrow v \qquad s_2, E \Downarrow E'}{if\, e\, s_1\, s_2, E \Downarrow E'} \quad v = 0$$

Let's just add ♯ everywhere!

$$\textsc{IfAdHoc}\quad \frac{e, E^\sharp \Downarrow^\sharp v^\sharp \qquad s_1, E^\sharp \Downarrow^\sharp E_1^\sharp \qquad s_2, E^\sharp \Downarrow^\sharp E_2^\sharp}{if\, e\, s, E^\sharp \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp} \quad v^\sharp = \top$$

1. Motivation
2. Pretty-Big-Step: a Generic Rule Format
3. Defining an Abstract Semantics Correct by Construction
4. Running Abstract Interpreters

Pretty-Big-Step

- Introduced by Chargéraud (ESOP 2013).
- Can be compiled from Small-Step (ESOP 2014).
- Similar to Big-Step semantics.
- But much more constrainted.

Pretty-Big-Step

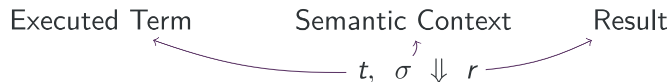
- Introduced by Chargéraud (ESOP 2013).
- Can be compiled from Small-Step (ESOP 2014).
- Similar to Big-Step semantics.
- But much more constrained.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow ax(\sigma)}\ cond(\sigma)$$

$$\textsc{Rule1}\quad \frac{\mathfrak{u}_1, up(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r}\ cond(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, up(\sigma) \Downarrow r \qquad \mathfrak{n}_2, next(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'}\ cond(\sigma)$$

Pretty-Big-Step



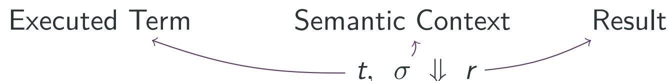
Each rule has

- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow ax(\sigma)}\ cond(\sigma) \qquad \textsc{Rule1}\quad \frac{\mathfrak{u}_1, up(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r}\ cond(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, up(\sigma) \Downarrow r \qquad \mathfrak{n}_2, next(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'}\ cond(\sigma)$$

Pretty-Big-Step



Each rule has

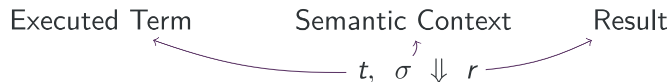
- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow ax(\sigma)} \quad cond(\sigma)$$

$$\textsc{Rule1}\quad \frac{\mathfrak{u}_1, up(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r} \quad cond(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, up(\sigma) \Downarrow r \qquad \mathfrak{n}_2, next(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'} \quad cond(\sigma)$$

Pretty-Big-Step



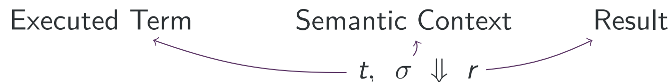
Each rule has

- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow ax(\sigma)} \quad cond(\sigma) \qquad\qquad \textsc{Rule1}\quad \frac{\mathfrak{u}_1, up(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r} \quad cond(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, up(\sigma) \Downarrow r \qquad \mathfrak{n}_2, next(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'} \quad cond(\sigma)$$

Pretty-Big-Step



Each rule has

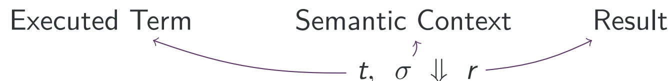
- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow ax(\sigma)} \quad cond(\sigma)$$

$$\textsc{Rule1}\quad \frac{\mathfrak{u}_1, up(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r} \quad cond(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, up(\sigma) \Downarrow r \qquad \mathfrak{n}_2, next(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'} \quad cond(\sigma)$$

Pretty-Big-Step



Each rule has

- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

$$\textsc{Axiom}\quad \frac{}{\mathfrak{l}, \sigma \Downarrow \mathit{ax}(\sigma)} \quad \mathit{cond}(\sigma)$$

$$\textsc{Rule1}\quad \frac{\mathfrak{u}_1, \mathit{up}(\sigma) \Downarrow r}{\mathfrak{l}, \sigma \Downarrow r} \quad \mathit{cond}(\sigma)$$

$$\textsc{Rule2}\quad \frac{\mathfrak{u}_2, \mathit{up}(\sigma) \Downarrow r \qquad \mathfrak{n}_2, \mathit{next}(\sigma, r) \Downarrow r'}{\mathfrak{l}, \sigma \Downarrow r'} \quad \mathit{cond}(\sigma)$$

Abstract Rules

Each rule has

- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

Abstract Rules

Shared between the concrete and abstract semantics

Each rule has

- A structural part: identifier, terms;
- A semantic part: side-conditions, transfer functions.

To be specified in the abstract semantics.
To be *locally* proved correct.

- The abstract semantics will follow the exact same structure as the concrete semantics.

Abstract Semantics

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

Concrete Semantics $\Downarrow$

$$\overline{}$$

At each step,
apply *one* rule that applies

Abstract Semantics $\Downarrow^\sharp$

$$\overline{}$$

At each step,
apply *all* the rules that apply

Abstract Semantics

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

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

Abstract Semantics

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

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

Example of Concrete Rules

$$\textsc{While}(e,s)\quad \frac{\mathit{while}_1\ e\,s,\ \mathit{ret}\ E \Downarrow o}{\mathit{while}\ e\,s,\ E \Downarrow o}$$

$$\textsc{While1}(e,s)\quad \frac{e, E \Downarrow o \qquad \mathit{while}_2\ e\,s,\ (E,o) \Downarrow o'}{\mathit{while}_1\ e\,s,\ \mathit{ret}\ E \Downarrow o'}$$

$$\textsc{While2True}(e,s)\quad \frac{s, E \Downarrow o \qquad \mathit{while}_1\ e\,s,\ o \Downarrow o'}{\mathit{while}_2\ e\,s,\ (E, \mathit{val}\ v) \Downarrow o'}\quad v \neq 0$$

$$\textsc{While2False}(e,s)\quad \frac{}{\mathit{while}_2\ e\,s,\ (E, \mathit{val}\ v) \Downarrow \mathit{ret}\ E}\quad v = 0$$

Example of Abstract Rules

$$\textsc{While}(e, s)\quad \frac{\mathit{while}_1\ e\,s, E^\sharp \Downarrow^\sharp o^\sharp}{\mathit{while}\ e\,s, E^\sharp \Downarrow^\sharp o^\sharp}$$

$$\textsc{While1}(e, s)\quad \frac{e, E^\sharp \Downarrow^\sharp v^\sharp \qquad \mathit{while}_2\ e\,s, (E^\sharp, v^\sharp) \Downarrow^\sharp o^\sharp}{\mathit{while}_1\ e\,s, E^\sharp \Downarrow^\sharp o^\sharp}$$

$$\textsc{While2True}(e, s)\quad \frac{s, E^\sharp \Downarrow^\sharp o \qquad \mathit{while}_1\ e\,s, o^\sharp \Downarrow^\sharp o'^\sharp}{\mathit{while}_2\ e\,s, (E^\sharp, v^\sharp) \Downarrow^\sharp o'^\sharp}\quad \gamma\left(v^\sharp\right) \cap \mathbb{Z}^\star \neq \emptyset$$

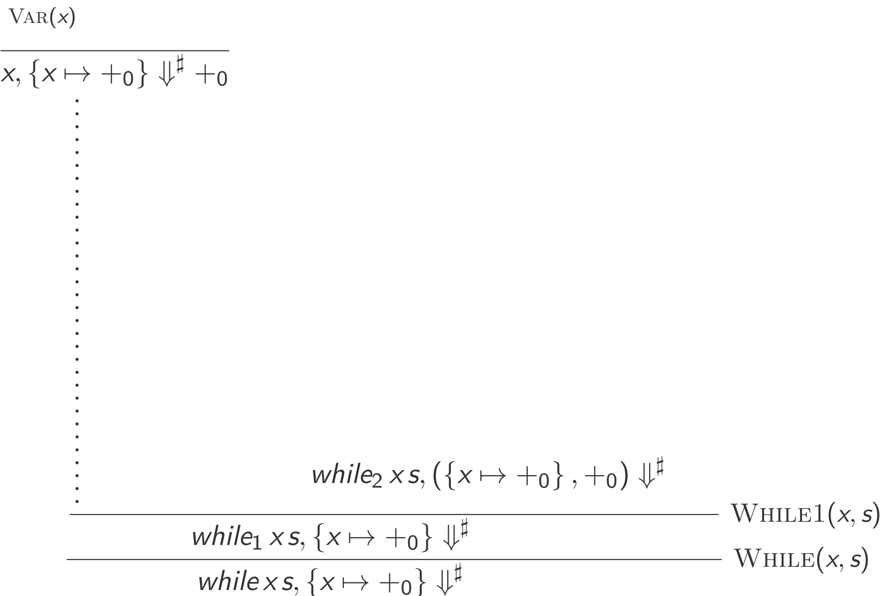
$$\textsc{While2False}(e, s)\quad \frac{}{\mathit{while}_2\ e\,s, (E^\sharp, v^\sharp) \Downarrow^\sharp E^\sharp}\quad 0 \in \gamma\left(v^\sharp\right)$$

Example of Abstract Derivation Tree

$$\text{While1}(e, s)\ \frac{e, E \Downarrow o \qquad while_2\ e\, s, (E, o) \Downarrow o'}{while_1\ e\, s, ret\ E \Downarrow o'}$$

$$\dfrac{\dfrac{}{while_1\ x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \text{While1}(x, s)}{while\ x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \text{While}(x, s)$$

Example of Abstract Derivation Tree



Example of Abstract Derivation Tree

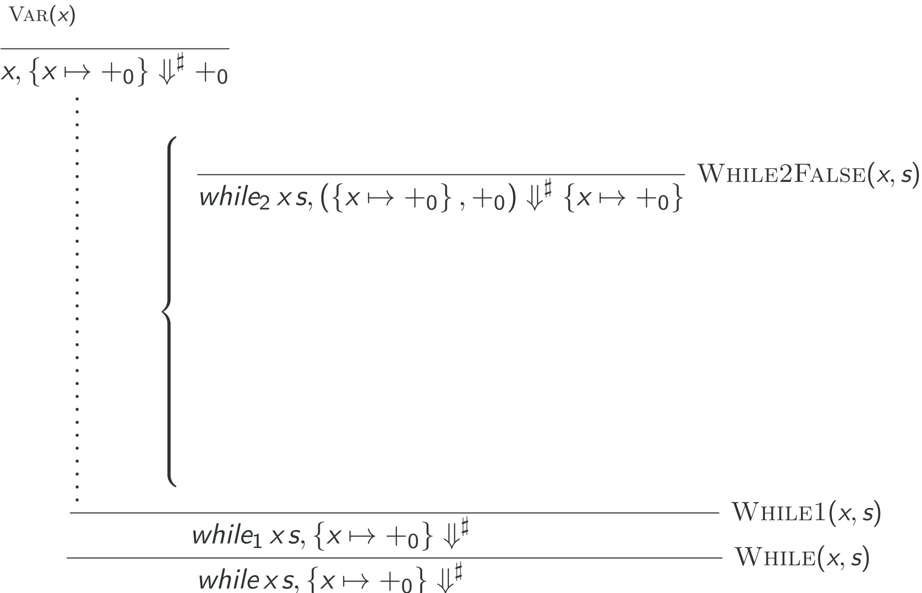
$$\dfrac{\textsc{Var}(x)}{x, \{x \mapsto +_0\} \Downarrow^\sharp +_0}$$

$$\textsc{While2True}(e, s)\quad \dfrac{s, E^\sharp \Downarrow^\sharp o \qquad while_1\; e\, s, o^\sharp \Downarrow^\sharp o'^\sharp}{while_2\; e\, s, (E^\sharp, v^\sharp) \Downarrow^\sharp o'^\sharp} \quad \gamma\left(v^\sharp\right) \cap \mathbb{Z}^\star \neq \emptyset$$

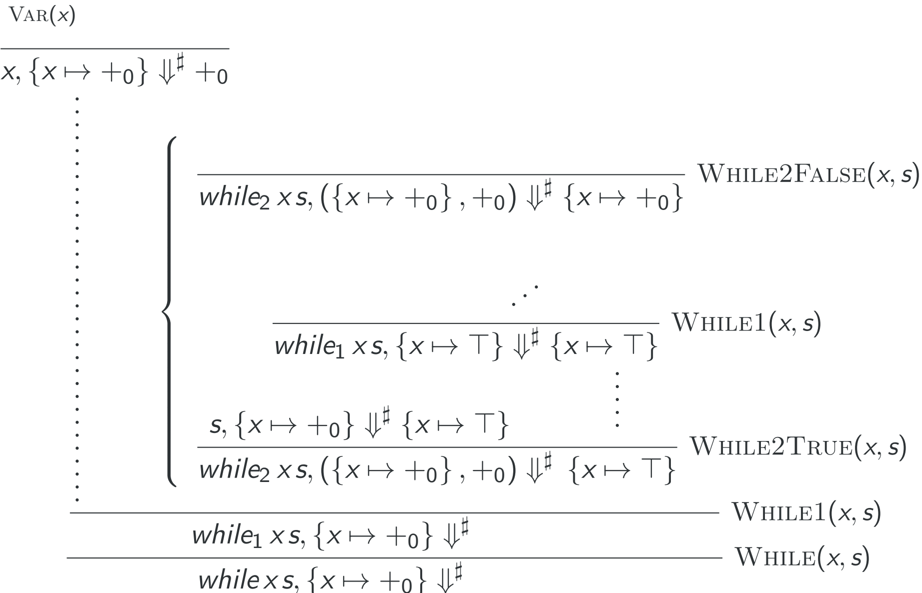
$$\textsc{While2False}(e, s)\quad \dfrac{}{while_2\; e\, s, (E^\sharp, v^\sharp) \Downarrow^\sharp E^\sharp} \quad 0 \in \gamma\left(v^\sharp\right)$$

$$\dfrac{\dfrac{\vdots \qquad while_2\; x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp}{while_1\; x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While1}(x, s)}{while\; x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While}(x, s)$$

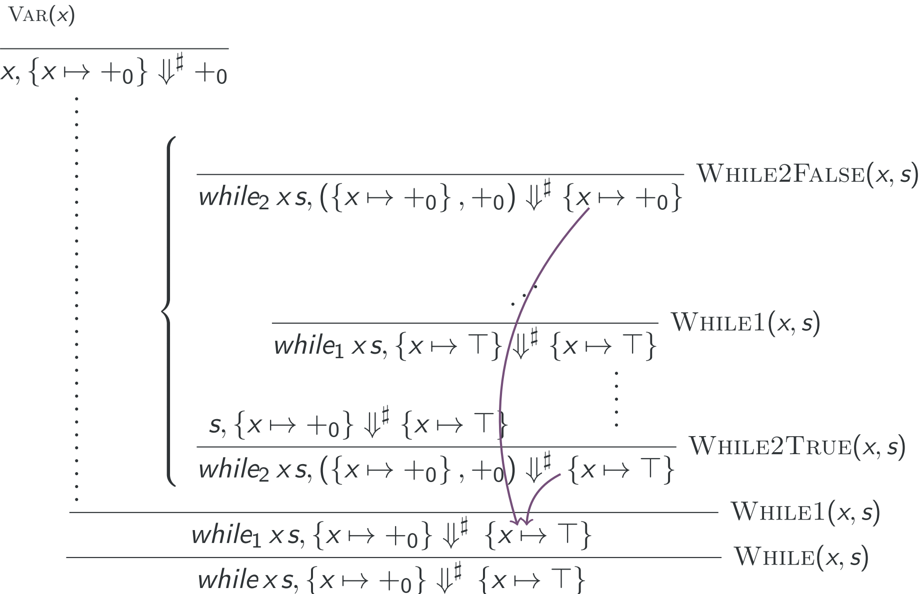
Example of Abstract Derivation Tree



Example of Abstract Derivation Tree



Example of Abstract Derivation Tree



An Abstract Semantics Correct by Construction

Hypotheses:

- Correctness of the side-conditions,
- Correctness of the transfer functions.

Theorem (Correctness)

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

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

An Abstract Semantics Correct by Construction

Hypotheses:

- Correctness of the side-conditions,
- Correctness of the transfer functions.

Theorem (Correctness)

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

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

Proven independently of the rules!

Concrete Domains

`int`, `bool`

Concrete Operations

`+`, `=`

Concrete Semantics

$t, \sigma \Downarrow r$

Abstract Domains

Sign

Abstract Operations

$+^\sharp$, $=^\sharp$

Abstract Semantics

$t, \sigma^\sharp \Downarrow^\sharp r^\sharp$

Abstract Interpreter

$f\left(t, \sigma^\sharp\right) = r^\sharp$

Defining Abstract Interpreters: a Verifier

- An abstract interpreter is a function building an abstract derivation.
- But this abstract semantic tree can be infinite!

A Verifier

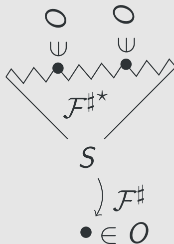
- It takes an oracle, i.e., a set O of triples $t, \sigma^\sharp, r^\sharp$.

It tries to prove $O \subseteq {\mathcal{F}^\sharp}^+(O)$.
By PARK's principle, this implies $O \subseteq \Downarrow^\sharp$.

- An abstract interpreter is a function building an abstract derivation.
- But this abstract semantic tree can be infinite!

A Verifier

- It takes an oracle, i.e., a set O of triples $t, \sigma^\sharp, r^\sharp$.



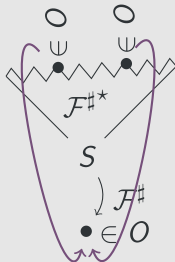
It tries to prove $O \subseteq \mathcal{F}^{\sharp^+}(O)$.
By PARK's principle, this implies $O \subseteq \Downarrow^\sharp$.

Defining Abstract Interpreters: a Verifier

- An abstract interpreter is a function building an abstract derivation.
- But this abstract semantic tree can be infinite!

A Verifier

- It takes an oracle, i.e., a set O of triples $t, \sigma^\sharp, r^\sharp$.



It tries to prove $O \subseteq \mathcal{F}^{\sharp^+}(O)$.
By PARK's principle, this implies $O \subseteq \Downarrow^\sharp$.

Generic Abstract Interpreters

- We have built some *generic* abstract interpreters.
- We can extract them to OCaml and run them.

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

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

Generic Abstract Interpreters

- We have built some *generic* abstract interpreters.
- We can extract them to OCaml and run them.

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

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

Generic Abstract Interpreters

- We have built some *generic* abstract interpreters.
- We can extract them to OCaml and run them.

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

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

Conclusion and Future Works

We have investigated how to define, in Coq, certified abstract interpreters for pretty-big-step semantics.

Recipe

1. define the concrete semantics;
2. define the abstract domains and operations on the abstract domain,
   - this automatically defines an abstract semantics;
3. prove the abstract operations are correct,
   - this implies the abstract semantics is correct;
4. define an analysis.

Future Works

- Apply it to JSCert.
- Allow non-local reasonning.
- Taking into account non-terminating behaviours.

Thanks You for Listening!

Concrete Domains

`int`, `bool`

Concrete Operations

`+`, `=`

Concrete Semantics

$t, \sigma \Downarrow r$

Abstract Domains

Sign

Abstract Operations

$+^\sharp$, $=^\sharp$

Abstract Semantics

$t, \sigma^\sharp \Downarrow^\sharp r^\sharp$

Abstract Interpreter

$f\left(t, \sigma^\sharp\right) = r^\sharp$

Bonus Slides

Concrete Semantics

$$
\begin{array}{l}
apply_i\,(\Downarrow_0) := \\
\quad match\ rule(i)\ \ with \\
\quad \mid\ \ Ax\,(ax) \qquad\quad \Rightarrow \{(\mathfrak{l}_i, \sigma, r) \mid ax(\sigma) = \texttt{Some}(r)\} \\
\quad \mid\ \ R_1\,(up) \qquad\quad \Rightarrow \left\{ (\mathfrak{l}_i, \sigma, r) \ \middle|\ \begin{array}{l} \quad up\,(\sigma) = \texttt{Some}(\sigma') \\ \wedge\, \mathfrak{u}_{1,i}, \sigma' \Downarrow_0 r \end{array} \right\} \\
\quad \mid\ \ R_2\,(up, next) \Rightarrow \left\{ (\mathfrak{l}_i, \sigma, r) \ \middle|\ \begin{array}{l} \quad up\,(\sigma) = \texttt{Some}(\sigma') \\ \wedge\, \mathfrak{u}_{2,i}, \sigma' \Downarrow_0 r_1 \\ \wedge\, next\,(\sigma, r_1) = \texttt{Some}(\sigma'') \\ \wedge\, \mathfrak{n}_{2,i}, \sigma'' \Downarrow_0 \texttt{Some}(r) \end{array} \right\}
\end{array}
$$

$\Downarrow = lfp\,(\mathcal{F})$

$\mathcal{F}\,(\Downarrow_0) = \{(t, \sigma, r) \mid \exists i, cond_i\,(\sigma) \wedge (t, \sigma, r) \in apply_i\,(\Downarrow_0)\}$

Abstract Semantics

$$apply_i^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (t, \sigma, r) \;\middle|\; \begin{array}{l} \exists \sigma_0, \exists r_0, \\ \quad \sigma \sqsubseteq^\sharp \sigma_0 \wedge r_0 \sqsubseteq^\sharp r \wedge \\ \quad (t, \sigma_0, r_0) \in apply_i\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$\Downarrow^\sharp = gfp\left(\mathcal{F}^\sharp\right)$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (t, \sigma, r) \;\middle|\; \begin{array}{l} \forall i.\ t = \mathfrak{l}_i \Rightarrow cond_i(\sigma) \Rightarrow \\ \qquad (t, \sigma, r) \in apply_i^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

Non Local Reasonning

$$if\,x\ (r := 0)\ (r := x)$$

Analysing in $\{x \mapsto +\}$

- Only the rule IfTrue applies.
- We get $r \mapsto 0$.

Analysing in $\{x \mapsto \top\}$

- Both rules IfTrue and IfFalse apply.
- We get $r \mapsto 0$ from IfTrue.
- We get $r \mapsto \top$ from IfFalse.
- We get $r \mapsto \top$ at the end.

```
CoInductive aeval : term -> ast -> ares -> Prop :=
  | aeval_cons : forall t sigma r,
      (forall n,
        t = left n ->
        acond n sigma ->
        aapply n sigma r) ->
      aeval t sigma r
with aapply : name -> ast -> ares -> Prop :=
  | aapply_cons : forall n sigma sigma' r r',
      sigma ⊑ sigma' ->
      r' ⊑ r ->
      aapply_step n sigma' r' ->
      aapply n sigma r
```

```
with aapply_step : name -> ast -> ares -> Prop :=
  | aapply_step_Ax : forall n ax sigma r,
      rule_struct n = Rule_struct_Ax _ ->
      arule n = Rule_Ax ax ->
      ax sigma = Some r ->
      aapply_step n sigma r
  | aapply_step_R1 : forall n t up sigma sigma' r,
      rule_struct n = Rule_struct_R1 t ->
      arule n = Rule_R1 _ up ->
      up sigma = Some sigma' ->
      aeval t sigma' r ->
      aapply_step n sigma r
  | aapply_step_R2 : forall n t1 t2 up next
                        sigma sigma1 sigma2 r r',
      rule_struct n = Rule_struct_R2 t1 t2 ->
      arule n = Rule_R2 up next ->
      up sigma = Some sigma1 ->
      aeval t1 sigma1 r ->
      next sigma r = Some sigma2 ->
      aeval t2 sigma2 r' ->
      aapply_step n sigma r'.
```

1. Motivation
2. Pretty-Big-Step: a Generic Rule Format
3. Defining an Abstract Semantics Correct by Construction
4. Running Abstract Interpreters