Certified Abstract Interpretation with Pretty-Big-Step Semantics

Martin Bodin   Thomas Jensen   Alan Schmitt

Inria

13th 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?

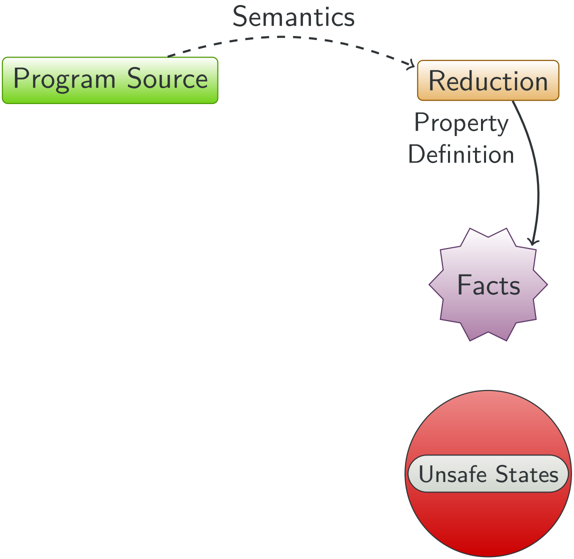
... proven in Coq?

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

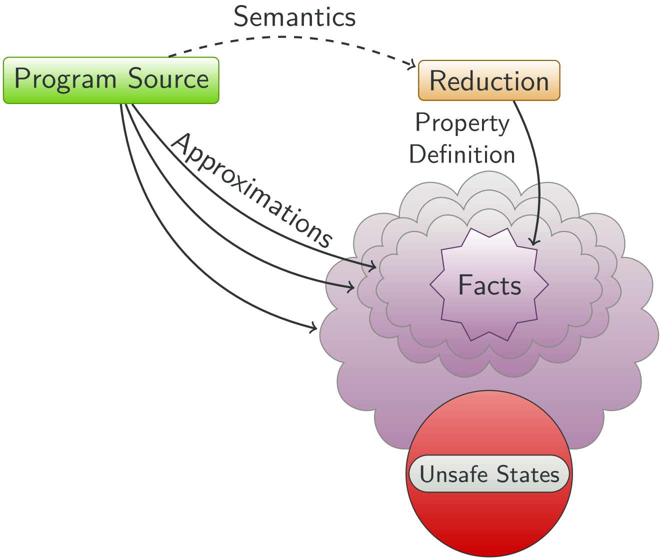
... proven in Coq?

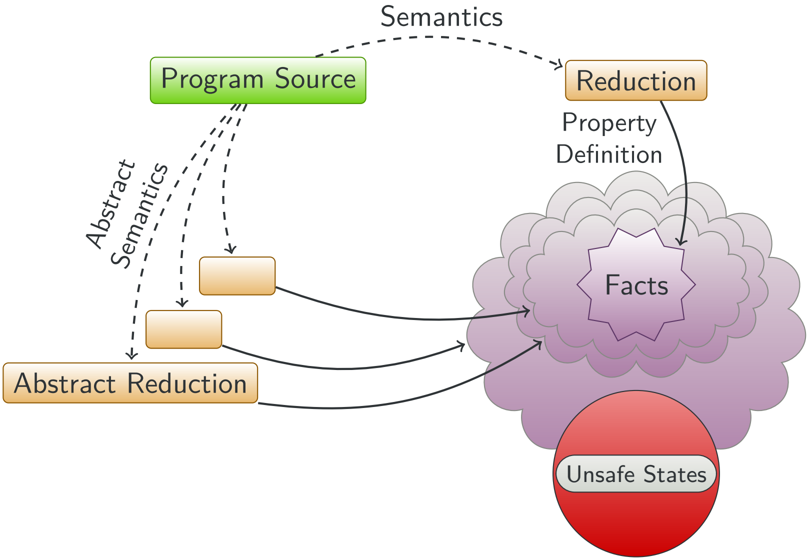
How to avoid *ad-hoc* abstract rules?

Abstract Interpretation

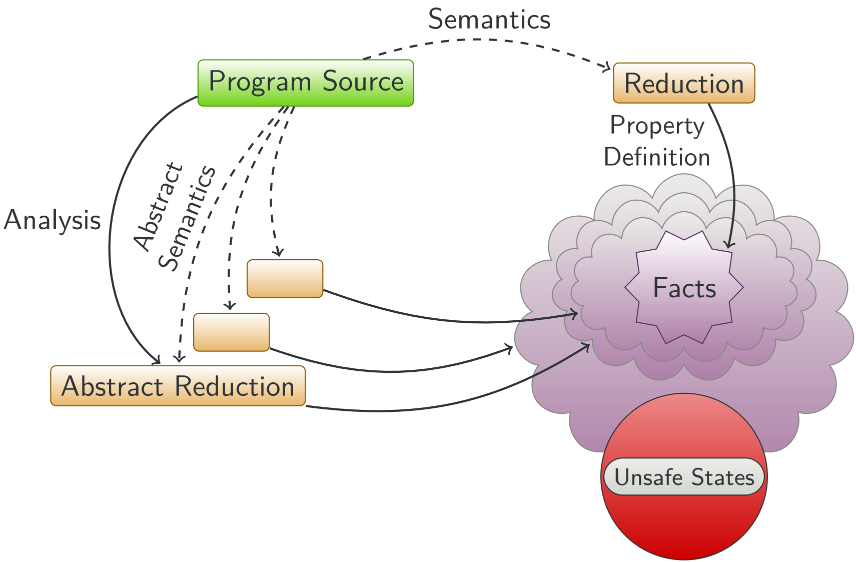


Abstract Interpretation




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

Abstract Interpretation



General Approach

Inspired by SCHMIDT's works:

Natural-Semantics-Based Abstract Interpretation (Preliminary Version)

$$\dfrac{\left\{\begin{array}{l}\dfrac{\vdots}{t_2', {\sigma_2^\sharp}' \Downarrow^\sharp {r^\sharp}_2'} \\[2em] \dfrac{}{t_2, \sigma_2^\sharp \Downarrow^\sharp r_2^\sharp}\end{array}\right. \qquad \dfrac{\dfrac{}{t_4, \sigma_4^\sharp \Downarrow^\sharp r_4^\sharp}}{t_3, \sigma_3^\sharp \Downarrow^\sharp r_3^\sharp}}{t_1, \sigma_1^\sharp \Downarrow^\sharp r_1^\sharp}$$

Abstract Derivation

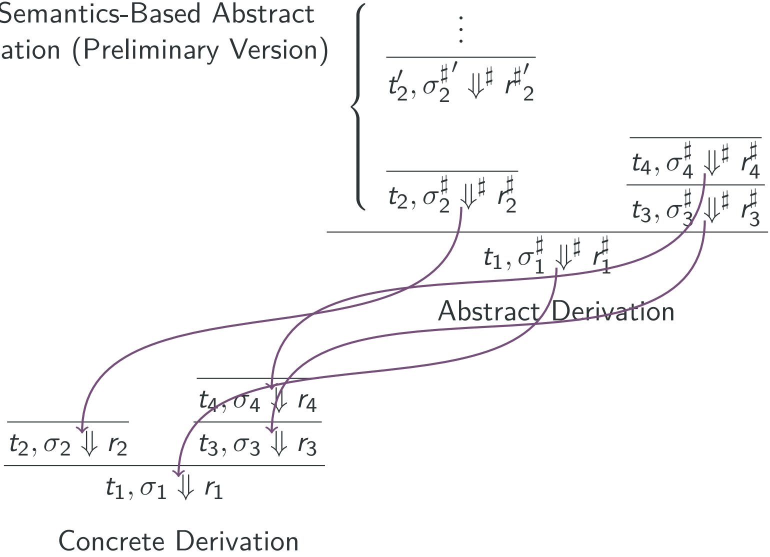
$$\dfrac{\dfrac{}{t_2, \sigma_2 \Downarrow r_2} \qquad \dfrac{\dfrac{}{t_4, \sigma_4 \Downarrow r_4}}{t_3, \sigma_3 \Downarrow r_3}}{t_1, \sigma_1 \Downarrow r_1}$$

Concrete Derivation

General Approach

Inspired by SCHMIDT's works:

Natural-Semantics-Based Abstract Interpretation (Preliminary Version)



Pretty-Big-Step

- Introduced by Charguéraud (ESOP 2013).
- Can be compiled from Small-Step (ESOP 2014).
- Similar to Big-Step semantics.

Pretty-Big-Step

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

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

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

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

Pretty-Big-Step

Executed Term — Semantic Context — Result

$$t,\ \sigma\ \Downarrow\ r$$

Each rule has

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

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

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

Pretty-Big-Step

Executed Term   Semantic Context   Result

$$t,\ \ \sigma\ \Downarrow\ r$$

Each rule has

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

Axiom

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

Rule1

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

Rule2

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

Pretty-Big-Step

Executed Term — Semantic Context — Result

$$t, \ \sigma \ \Downarrow \ r$$

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

Executed Term — Semantic Context — Result

$$t, \ \sigma \ \Downarrow \ r$$

Each rule has

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

Axiom

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

Rule1

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

Rule2

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

Pretty-Big-Step

Executed Term — Semantic Context — Result

$$t, \quad \sigma \quad \Downarrow \quad r$$

Each rule has

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

Axiom

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

Rule1

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

Rule2

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

1. Motivation

2. Pretty-Big-Step: a Generic Rule Format

3. Defining an Abstract Semantics Correct by Construction

4. Running Abstract Interpreters

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

$+^\sharp, =^\sharp$

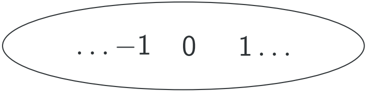
Abstract Semantics

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

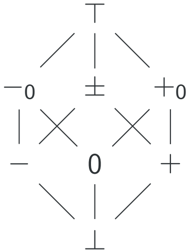
Abstract Interpreter

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

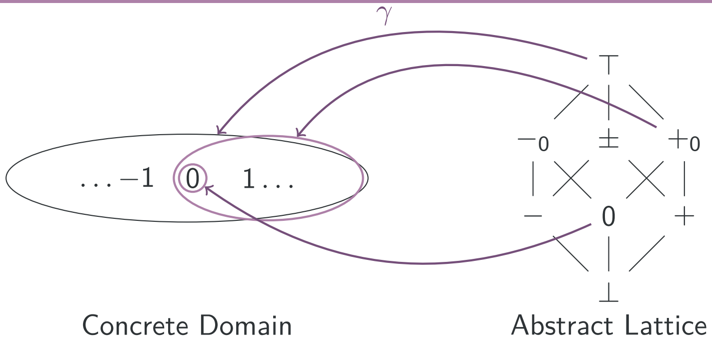
Defining Abstract Domains and Operations



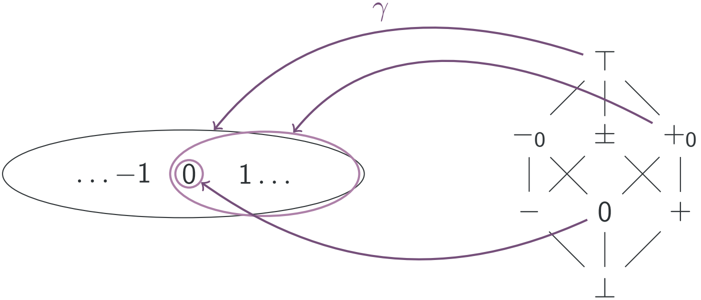
Concrete Domain



Abstract Lattice


γ
$\top$
$-_0$
$\pm$
$+_0$
$\ldots -1$
0
$1 \ldots$
$-$
0
$+$
$\bot$
Concrete Domain
Abstract Lattice

Defining Abstract Domains and Operations

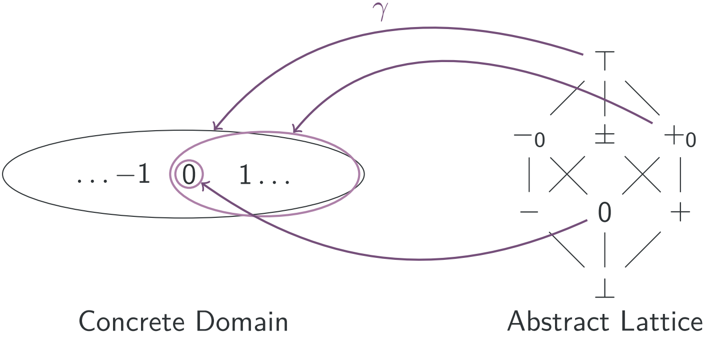


Concrete Domain

Abstract Lattice

$\cdot +^\sharp \cdot$	$\bot$	$-$	0	$+$	$-_0$	$\pm$	$+_0$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$-$	$\bot$	$-$	$-$	$\top$	$-$	$\top$	$\top$	$\top$
0	$\bot$	$-$	0	$+$	$-_0$	$\pm$	$+_0$	$\top$
$+$	$\bot$	$\top$	$+$	$+$	$\top$	$\top$	$+$	$\top$
$-_0$	$\bot$	$-$	$-_0$	$\top$	$-_0$	$\top$	$\top$	$\top$
$\pm$	$\bot$	$\top$	$\pm$	$\top$	$\top$	$\top$	$\top$	$\top$
$+_0$	$\bot$	$\top$	$+_0$	$+$	$\top$	$\top$	$+_0$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Defining Abstract Domains and Operations



The theory has already been formalized in Coq.

Cachera and Pichardie. A Certified Denotational Abstract Interpreter. *ITP'10*

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

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

$$\textsc{IfFalse}\ \frac{s_2, E \Downarrow E'}{if\ s_1\ s_2, (v, E) \Downarrow E'}\quad v \in \{0\}$$

Defining an Abstract Semantics, the Direct Approach

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

Let's just add ♯ everywhere!

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

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

Defining an Abstract Semantics, the Direct Approach

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

$$\textsc{IfFalse}\quad \frac{s_2, E \Downarrow E'}{\mathit{if}\ s_1\ s_2, (v, E) \Downarrow E'}\ v \in \{0\}$$

Let's just add $\sharp$ everywhere!

$$\textsc{IfAdHoc}\quad \frac{s_1, E^\sharp \Downarrow^\sharp E_1^\sharp \qquad s_2, E^\sharp \Downarrow^\sharp E_2^\sharp}{\mathit{if}\ s_1\ s_2, \left(v^\sharp, E^\sharp\right) \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}\ v^\sharp = \top$$

Abstract Rules

In pretty-big-step, each rule has

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

Abstract Rules

Shared between the concrete and abstract semantics

In pretty-big-step, 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 $\Downarrow$	Abstract Semantics $\Downarrow^\sharp$
At each step, apply *one* rule that applies	At each step, apply *all* the rules that apply

$$\dfrac{\begin{array}{cc} s_1, E_0^\sharp \Downarrow E_1^\sharp & s_2, E_0^\sharp \Downarrow E_2^\sharp \\ \uparrow \textsc{IfTrue} & \uparrow \textsc{IfFalse} \end{array}}{if\ s_1\ s_2, \left(v^\sharp, E_0^\sharp\right) \Downarrow E_1^\sharp \sqcup E_2^\sharp}$$

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
	Allow approximations

$$\dfrac{s_1, E_0^\sharp \Downarrow E_1^\sharp \;\uparrow \text{IfTrue} \qquad s_2, E_0^\sharp \Downarrow E_2^\sharp \;\uparrow \text{IfFalse}}{\mathit{if}\ s_1\ s_2, \left(v^\sharp, E_0^\sharp\right) \Downarrow E_1^\sharp \sqcup E_2^\sharp}$$

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
	Allow approximations
Inductive interpretation of the rules $\Downarrow = lfp\,(\mathcal{F})$	Co-inductive interpretation of the rules $\Downarrow^\sharp = gfp\left(\mathcal{F}^\sharp\right)$

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

Example of Concrete Rules

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

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

$$\textsc{While2True}(e, s)\quad \frac{s, E \Downarrow o \qquad \mathit{while}_1\ e\, s, o \Downarrow o'}{\mathit{while}_2\ e\, s, (E, \mathit{val}\ v) \Downarrow o'} \quad v \in \mathbb{Z}^\star$$

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

Example of a Concrete Derivation Tree

$s = (x := x - 1)$

$$\frac{\textsc{Var}(x)}{x, \{x \mapsto 1\} \Downarrow 1}$$

$$\frac{\textsc{Var}(x)}{x, \{x \mapsto 0\} \Downarrow 0}$$

$$\frac{\vdots}{\mathit{while}_2\, x\, s, (\{x \mapsto 0\}, \mathit{val}\, 0) \Downarrow \{x \mapsto 0\}}\ \textsc{While2False}(x, s)$$

$$\frac{\mathit{while}_2\, x\, s, (\{x \mapsto 0\}, \mathit{val}\, 0) \Downarrow \{x \mapsto 0\}}{\mathit{while}_1\, x\, s, \{x \mapsto 1\} \Downarrow \{x \mapsto 0\}}\ \textsc{While1}(x, s)$$

$$\frac{s, \{x \mapsto 1\} \Downarrow \{x \mapsto 0\} \qquad \vdots}{\mathit{while}_2\, x\, s, (\{x \mapsto 1\}, \mathit{val}\, 1) \Downarrow \{x \mapsto 0\}}\ \textsc{While2True}(x, s)$$

$$\frac{\vdots \qquad \mathit{while}_2\, x\, s, (\{x \mapsto 1\}, \mathit{val}\, 1) \Downarrow \{x \mapsto 0\}}{\mathit{while}_1\, x\, s, \{x \mapsto 1\} \Downarrow \{x \mapsto 0\}}\ \textsc{While1}(x, s)$$

$$\frac{\mathit{while}_1\, x\, s, \{x \mapsto 1\} \Downarrow \{x \mapsto 0\}}{\mathit{while}\, x\, s, \{x \mapsto 1\} \Downarrow \{x \mapsto 0\}}\ \textsc{While}(x, s)$$

Example of Abstract Rules

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

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

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

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

Example of an Abstract Derivation Tree

$s = (x := x - 1)$

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

$$\cfrac{\cfrac{}{\textit{while}_1\; x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\;\textsc{While1}(x, s)}{\textit{while}\; x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\;\textsc{While}(x, s)$$

Example of an Abstract Derivation Tree

$s = (x := x - 1)$

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

$$\dfrac{\dfrac{\vdots \quad \textit{while}_2\ x\ s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp}{\textit{while}_1\ x\ s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While1}(x, s)}{\textit{while}\ x\ s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While}(x, s)$$

Example of an Abstract Derivation Tree

$s = (x := x - 1)$

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

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

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

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

Example of an Abstract Derivation Tree

$s = (x := x - 1)$

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

$\vdots$

$$\dfrac{}{while_2\, x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp \{x \mapsto +_0\}}\ \textsc{While2False}(x, s)$$

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

Example of an Abstract Derivation Tree

$$s = (x := x - 1)$$

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

$$\vdots$$

$$\left\{ \begin{array}{l}
\dfrac{}{\mathit{while}_2\, x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp \{x \mapsto +_0\}}\ \textsc{While2False}(x, s) \\[2em]
\dfrac{\iddots}{\mathit{while}_1\, x\, s, \{x \mapsto \top\} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While1}(x, s) \\
\qquad\qquad\qquad\qquad\qquad\qquad\vdots \\
\dfrac{s, \{x \mapsto +_0\} \Downarrow^\sharp \{x \mapsto \top\}}{\mathit{while}_2\, x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While2True}(x, s)
\end{array} \right.$$

$$\dfrac{\dfrac{}{\mathit{while}_1\, x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While1}(x, s)}{\mathit{while}\, x\, s, \{x \mapsto +_0\} \Downarrow^\sharp}\ \textsc{While}(x, s)$$

Example of an Abstract Derivation Tree

$s = (x := x - 1)$

$\textsc{Var}(x)$

$$\overline{x, \{x \mapsto +_0\} \Downarrow^\sharp +_0}$$

$$\overline{\mathit{while}_2\ x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp \{x \mapsto +_0\}}\ \textsc{While2False}(x, s)$$

$$\overline{\mathit{while}_1\ x\, s, \{x \mapsto \top\} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While1}(x, s)$$

$$\frac{s, \{x \mapsto +_0\} \Downarrow^\sharp \{x \mapsto \top\}}{\mathit{while}_2\ x\, s, (\{x \mapsto +_0\}, +_0) \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While2True}(x, s)$$

$$\overline{\mathit{while}_1\ x\, s, \{x \mapsto +_0\} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While1}(x, s)$$

$$\overline{\mathit{while}\ x\, s, \{x \mapsto +_0\} \Downarrow^\sharp \{x \mapsto \top\}}\ \textsc{While}(x, s)$$

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.

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

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.

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

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

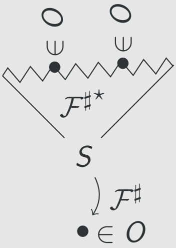
- An abstract interpreter is a function building an abstract derivation.

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



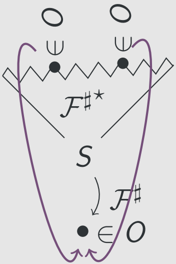
It tries to prove $O \subseteq {\mathcal{F}^\sharp}^+(O)$.
By PARK's principle, this implies $O \subseteq \Downarrow^\sharp$.

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.

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

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

- 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

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

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

$$\mathcal{F}\,(\Downarrow_0) = \{(t, \sigma, r) \mid \exists i, \mathit{cond}_i\,(\sigma) \wedge (t, \sigma, r) \in \mathit{apply}_i\,(\Downarrow_0)\}$$

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 Reasoning

$$\mathit{if}\, x\ (r := 0)\ (r := x)$$

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