Deriving Abstract Interpreters from Skeletal Semantics

Thomas Jensen, **Vincent Rébiscoul**, Alan Schmitt

September 18, 2023

INRIA, Université de Rennes

Introduction

Motivation

- Code runs in critical applications
- Ensuring that software works without errors and as intended is important

Static Analyses: help keep your programs bug-free!

A **Static Analysis** checks some properties of a program without executing it

- There are static analyses fully automatic: abstract interpretation[1]
- Developing **correct** analyses is hard
- Issue: lots of pen an paper work

[1] Schmidt 1995.

Can a **correct** static analyser be **mechanically** derived from a language formal description?

Skeletal Semantics

A Framework to formalise Programming Language: Skeletal Semantics

- Skeletal Semantics[2] is a proposal for machine representable semantics
- A Skeletal Semantics is a description of a language
- **Skel** is the language of Skeletal Semantics: minimalist and functional
- The Necro Library[3] is a set of tools to manipulate Skeletal Semantics
  - Generate an OCaml interpreter
  - Coq description
  - This talk: **Static Analysis**

[2] Bodin et al. 2019.
[3] Noizet n.d.

$$
\begin{aligned}
\texttt{expr} \quad ::= \quad & \texttt{l} \in \texttt{lit} \\
\mid \quad & \texttt{x} \in \texttt{ident} \\
\mid \quad & \texttt{expr} + \texttt{expr} \\
\mid \quad & \texttt{expr} \leq \texttt{expr}
\end{aligned}
$$

```
(* Unspecified Types *)
type ident
type lit

(* Specified Types *)
type expr =
  | Const lit
  | Var ident
  | Plus (expr, expr)
  | Leq(expr, expr)
```

```
stmt  ::=  skip
       |   stmt ; stmt
       |   x := expr
       |   if expr then stmt else stmt
       |   while expr do stmt
```

```
type stmt =
  | Skip
  | Seq (stmt, stmt)
  | Assign (ident, expr)
  | If (expr, stmt, stmt)
  | While (expr, stmt)
```

$\mathtt{int} = \mathbb{Z}$

$\mathtt{lit} = \mathbb{Z}$

$\mathtt{store} = \mathtt{ident} \hookrightarrow \mathtt{int}$

$\Downarrow_{expr} \in \mathcal{P}\left((\mathtt{store} \times \mathtt{expr}) \times \mathtt{int}\right)$

$$\frac{c \in \mathtt{lit}}{\sigma, c \Downarrow_{expr} c} \qquad \frac{x \in \mathtt{ident}}{\sigma, x \Downarrow_{expr} \sigma(x)}$$

```
type store
type int
(* Unspecified terms *)
val litToInt : lit → int
val read : (ident, store) → int

(* Specified terms *)
val eval_expr ((s, e): (store, expr)): int =
  branch
    let Const i = e in
    litToInt i
  or
    let Var x = e in
    read (x, s)

...
```

$$\Downarrow_{stmt} \in \mathcal{P}\left((\mathtt{store} \times \mathtt{stmt}) \times \mathtt{store}\right)$$

$$\frac{\sigma, e \Downarrow_{expr} v \qquad v \neq 0 \qquad \sigma, s \Downarrow_{stmt} \sigma' \qquad \sigma', \mathit{while}\ e\ \mathit{do}\ s \Downarrow_{stmt} \sigma''}{\sigma, \mathit{while}\ e\ \mathit{do}\ s \Downarrow_{stmt} \sigma''}$$

$$\frac{\sigma, e \Downarrow_{expr} 0}{\sigma, \mathit{while}\ e\ \mathit{do}\ s \Downarrow_{stmt} \sigma}$$

```
val eval_stmt ((s, t): (store, stmt)): store =
  branch
    let While (cond, t') = t in
    let i = eval_expr (s, cond) in
    branch
      let () = isNotZero i in
      let s' = eval_stmt (s, t') in
      eval_stmt (s', t)
    or
      let () = isZero i in
      s
    end
    or
...
```

Skel Syntax : λ-calculus

$$
\begin{array}{llll}
\textsc{Term} & t & ::= & x \mid C\ t \mid (t, \ldots, t) \mid \lambda p : \tau \to S \\
\textsc{Skeleton} & S & ::= & t_0\ t_1 \ldots t_n \mid \textbf{let}\ p\ =\ S\ \textbf{in}\ S \mid (S..S) \mid t \\
\textsc{Type} & \tau & ::= & b \mid \tau \to \tau \mid (\tau, .., \tau)
\end{array}
$$

Abstract Interpretation: a framework to define static analyses

Abstract Interpretation

- Abstract Interpretation: method to define computable approximations of programs[4]
- **Goal:** cover all possible executions of a given program
- Example for WHILE: replace **relative integers** by **intervals**

$$\texttt{int} = \{[\![n_1, n_2]\!] \mid n_i \in \mathbb{Z} \cup \{+\infty, -\infty\}\}$$

$$\texttt{store} = \texttt{ident} \hookrightarrow \texttt{int}$$

[4] P. Cousot and R. Cousot 1977.

Abstract Interpretation of While

```
while^l0 x < 4 do
    x := x + 1^l1
done
```

Abstract Interpretation of While

$(l_0, In) : [x \mapsto [\![0, 0]\!]]$

```
while^l0 x < 4 do

    x := x + 1^l1

done
```

$\texttt{while}^{l_0}\ \texttt{x < 4 do}$

$\texttt{x := x + 1}^{l_1}$

Abstract Interpretation of While

$(l_0, In) : [x \mapsto [\![0, 0]\!]]$

while^{l_0} x < 4 do

$(l_1, In) : [x \mapsto [\![0, 0]\!]]$

x := x + 1^{l_1}

done

Abstract Interpretation of While

$(l_0, In) : [x \mapsto [\![0, 0]\!]]$

```
while^l0 x < 4 do
```

$\texttt{while}^{l_0}\ \texttt{x < 4 do}$

$(l_1, In) : [x \mapsto [\![0, 0]\!]]$

$\texttt{x := x + 1}^{l_1}$

$(l_1, Out) : [x \mapsto [\![1, 1]\!]]$

`done`

Abstract Interpretation of While

$(l_0, In) : [x \mapsto [\![0, 0]\!]]$

$\texttt{while}^{l_0}\ \texttt{x < 4 do}$

$(l_1, In) : [x \mapsto [\![0, 1]\!]]$

$\texttt{x := x + 1}^{l_1}$

$(l_1, Out) : [x \mapsto [\![1, 1]\!]]$

`done`

Abstract Interpretation of While

$(l_0, In) : [x \mapsto [\![0, 0]\!]]$

```
while^l0 x < 4 do
```

$(l_1, In) : [x \mapsto [\![0, 3]\!]]$

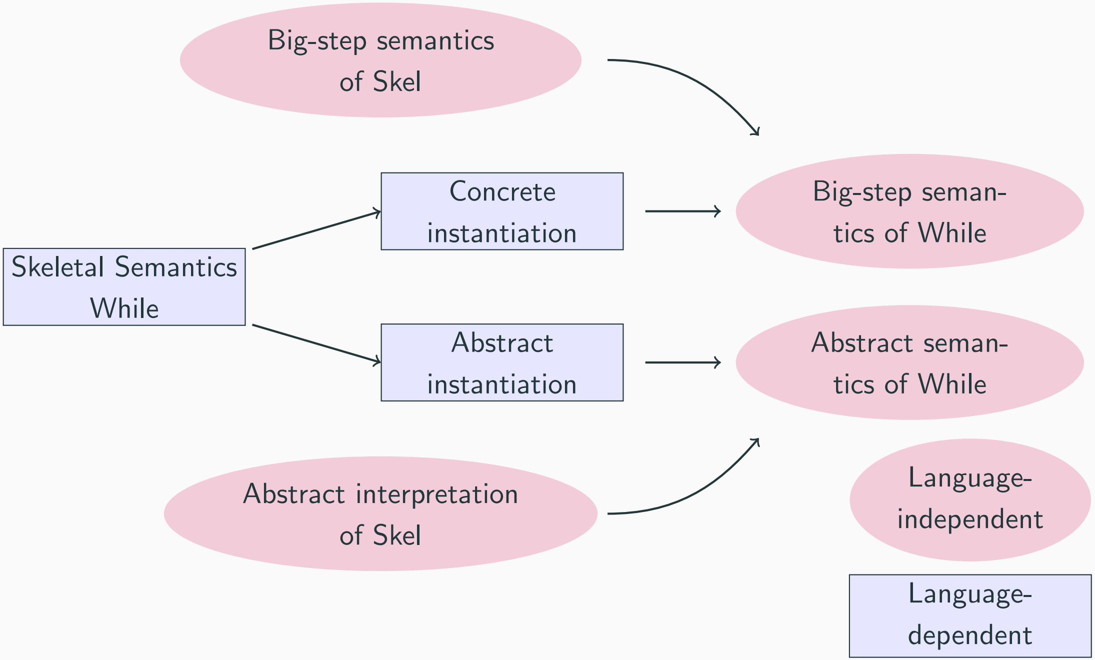
```
    x := x + 1^l1
```

$(l_1, Out) : [x \mapsto [\![1, 4]\!]]$

```
done
```

$(l_0, Out) : [x \mapsto [\![0, 4]\!]]$

From a Skeletal Semantics to an Abstract Interpretation


Big-step semantics of Skel
Skeletal Semantics While
Concrete instantiation
Big-step semantics of While
Abstract instantiation
Abstract semantics of While
Abstract interpretation of Skel
Language-independent
Language-dependent

Abstract Instantiation of While

Instantiation of Unspecified Types and Terms

$$V^\sharp(\mathit{ident}) \triangleq \mathcal{X}$$
$$V^\sharp(\mathit{lit}) \triangleq \mathbb{Z}$$
$$V^\sharp(\mathit{store}) \triangleq \mathcal{X} \hookrightarrow \mathbb{I}$$
$$V^\sharp(\mathit{int}) \triangleq \mathbb{I}$$

$$[\![\mathit{litToInt}]\!]^\sharp \triangleq \lambda i \to [\![i, i]\!]$$
$$[\![\mathit{read}]\!]^\sharp \triangleq \lambda(x, s^\sharp) \to s^\sharp(x)$$

$$[\![n_1, n_2]\!] \sqcup_{\mathtt{int}} [\![m_1, m_2]\!] = [\![\min(n_1, m_1), \max(n_2, m_2)]\!]$$
$$s_1^\sharp \sqcup_{\mathtt{store}} s_2^\sharp = \left[x \in \mathrm{dom}\ s_1^\sharp \cup \mathrm{dom}\ s_2^\sharp \mapsto s_1^\sharp(x) \sqcup_{\mathtt{int}} s_2^\sharp(x)\right]$$

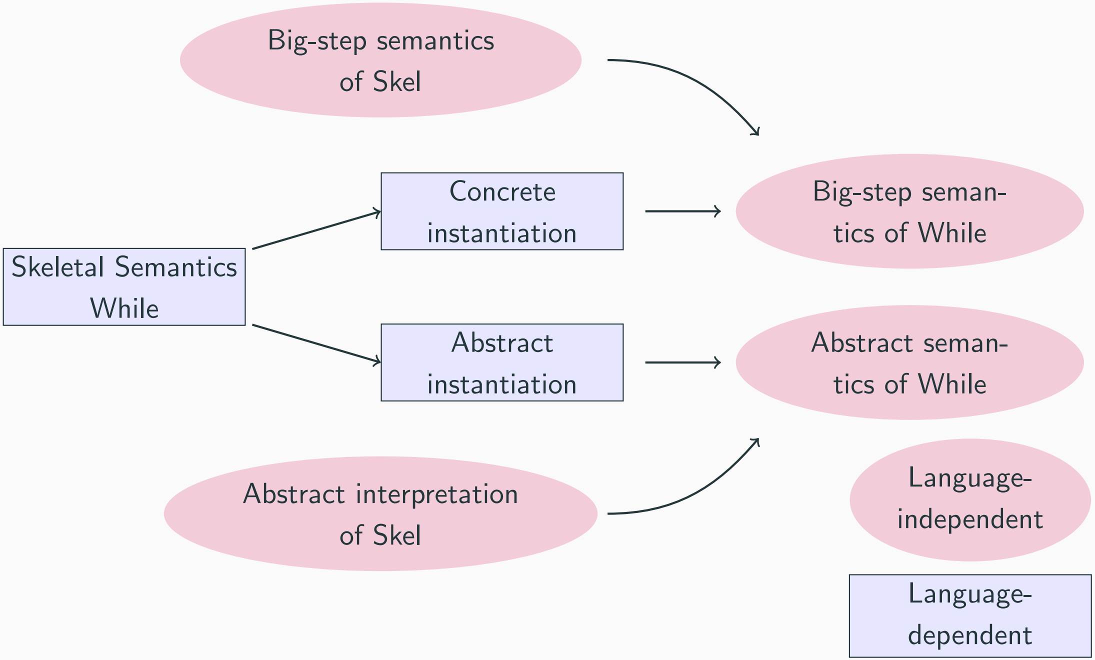
State of the Abstract Interpretation: $\mathcal{A}$

$\mathcal{A}$ is an AI-State

- Holds information collected throughout the abstract interpretation
- The state of the abstract interpretation only grows
- Language dependent

State of the Abstract Interpretation for While

$$\mathcal{A} \in \texttt{label} \times \{\,\textsc{In}, \textsc{Out}\,\} \to V^{\sharp}(\mathit{store})$$



Big-step semantics of Skel
Concrete instantiation
Big-step semantics of While
Skeletal Semantics While
Abstract instantiation
Abstract semantics of While
Abstract interpretation of Skel
Language-independent
Language-dependent

Instantiation of other types

The interpretation of the other types are automatically derived
Example: tuples

$$V^\sharp(\tau_1 \times .. \times \tau_n) = V^\sharp(\tau_1) \times .. \times V^\sharp(\tau_n)$$

$$(v_1^\sharp, .., v_n^\sharp) \sqcup_{\tau_1 \times .. \times \tau_n} (w_1^\sharp, .., w_n^\sharp) = (v_1^\sharp \sqcup_{\tau_1} w_1^\sharp, .., v_n^\sharp \sqcup_{\tau_n} w_n^\sharp)$$

Abstract Values and Environments

$$\textsc{AbstVal} = \bigcup_{\tau \in \textsc{Type}} V^\sharp(\tau) \qquad \textsc{AbstEnv} = \textsc{SkelVar} \hookrightarrow \textsc{AbstVal}$$

Abstract Interpretation of Skel[5]

$$\Downarrow^\sharp \in \mathcal{P}\left((\textsc{AIState} \times \textsc{AbstEnv} \times \textsc{Skeleton}) \times (\textsc{AbstVal} \times \textsc{AIState})\right)$$

$$\frac{\mathcal{A}, E^\sharp, S_i \Downarrow^\sharp v_i^\sharp, \mathcal{A}_i}{\mathcal{A}, E^\sharp, (S_1..S_n) \Downarrow^\sharp \sqcup^\sharp v_i^\sharp, \sqcup^\sharp \mathcal{A}_i} \textsc{Branch}$$

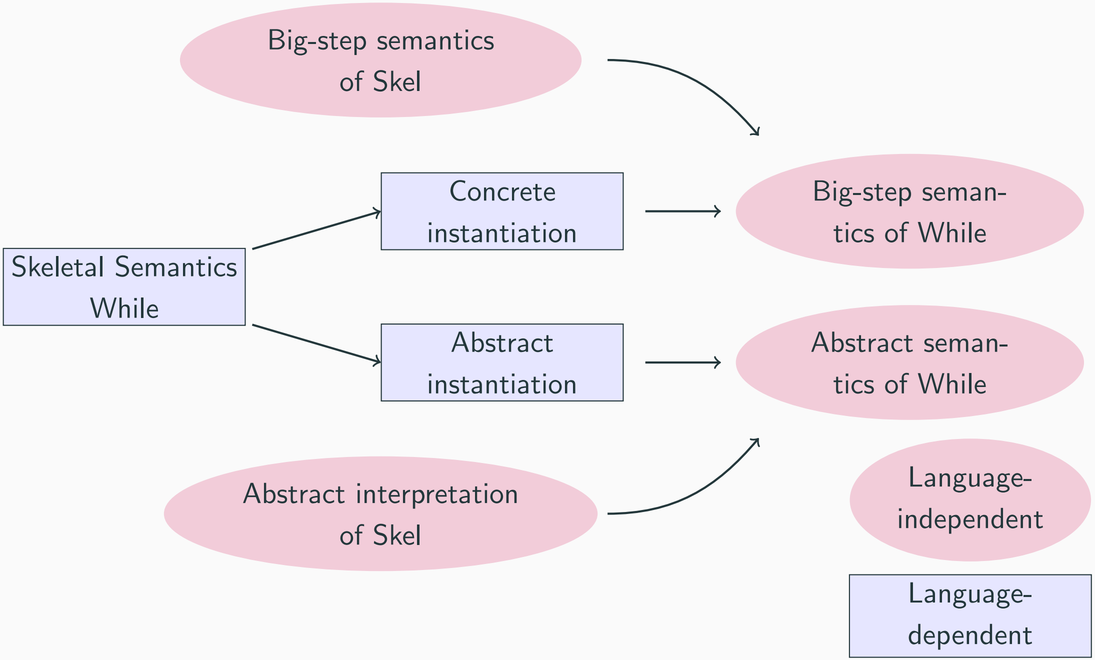
$$\frac{\mathcal{A}_0, E^\sharp, S_1 \Downarrow^\sharp v^\sharp, \mathcal{A}_1 \qquad \vdash E^\sharp + p \leftarrow v^\sharp \rightsquigarrow E'^\sharp \qquad \mathcal{A}_1, E'^\sharp, S_2 \Downarrow^\sharp w^\sharp, \mathcal{A}_2}{\mathcal{A}_0, E^\sharp, \mathsf{let}\ p = S_1\ \mathsf{in}\ S_2 \Downarrow^\sharp w^\sharp, \mathcal{A}_2} \textsc{LetIn}$$

[5] Schmidt 1995.

$$E_0^\sharp = \{\, s \mapsto [x \mapsto [\![0,0]\!]]\,,\, t \mapsto \mathit{While}(x \leq 3, x = x + 1)\,\}$$

$\mathcal{A}_0$ an empty AI-state

$$\mathcal{A}_0, E_0^\sharp, \mathit{eval\_st}\ (s,t) \Downarrow^\sharp \{\, x \mapsto [\![0,4]\!]\,\}\,,\mathcal{A}$$


Big-step semantics of Skel
Concrete instantiation
Big-step semantics of While
Skeletal Semantics While
Abstract instantiation
Abstract semantics of While
Abstract interpretation of Skel
Language-independent
Language-dependent

Concrete instantiation of While

Instantiation of Unspecified Types and Terms

$$V(ident) \triangleq \mathcal{X}$$
$$V(lit) \triangleq \mathbb{Z}$$
$$V(store) \triangleq \mathcal{X} \hookrightarrow \mathbb{Z}$$
$$V(int) \triangleq \mathbb{Z}$$

$$[\![litToInt]\!] \triangleq \lambda i \rightarrow i$$
$$[\![read]\!] \triangleq \lambda(x, s) \rightarrow s(x)$$

Big-step semantics of Skel

$$\frac{E, S_i \Downarrow_{sk} v_i}{E, (S_1..S_n) \Downarrow_{sk} v_i} \textsc{Branch}$$

$$\frac{E, S_1 \Downarrow_{sk} v \qquad \vdash E + p \leftarrow v \rightsquigarrow E' \qquad E', S_2 \Downarrow_{sk} w}{E, \text{let } p = S_1 \text{ in } S_2 \Downarrow^{\sharp} w} \textsc{LetIn}$$

The Correctness of the Abstract Interpretation

Concretisation functions

Definition
A **concretisation** function $\gamma_\tau : V^\sharp(\tau) \to \mathcal{P}\left(V(\tau)\right)$ maps an abstract value $v^\sharp$ to the set of concrete values it approximates.

Concretisation functions of the unspecified types

$$\gamma_{\texttt{int}}(\llbracket n_1, n_2 \rrbracket) \triangleq \llbracket n_1, n_2 \rrbracket \qquad \gamma_{\texttt{store}}(s^\sharp) \triangleq \left\{ s \;\middle|\; \forall x \in \mathrm{dom}\ s^\sharp,\ s(x) \in \gamma_{\texttt{int}}(s^\sharp(x)) \right\}$$

Correctness Theorem (simplified)

IF

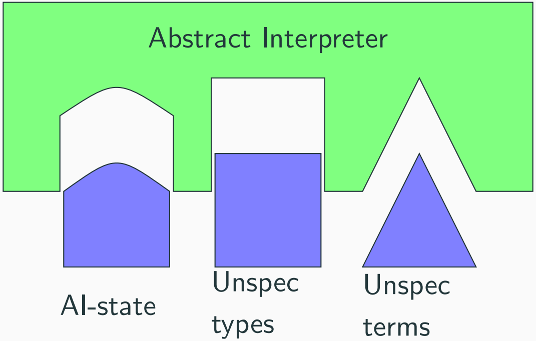
- The abstract instantiations of unspecified terms are correct approximation of the concrete instantiations of the unspecified terms
- Concretisation functions are monotonic

THEN

$$\left.\begin{array}{l} E \in \gamma(E^\sharp) \\ E, S \Downarrow_{sk} v \\ \mathcal{A}_0, E^\sharp, S \Downarrow_{sk} v^\sharp, \mathcal{A} \end{array}\right\} \implies v \in \gamma(v^\sharp)$$

Implementation[6]

- An abstract interpreter generator from a skeletal semantics
- Control Flow Analysis for λ-calculus, Interval Analysis for While



[6] Rébiscoul n.d.

Future Work

- Support for relational analyses
- Better interface for Abstract Interpreter Generator such that it is easier to use
- The abstract interpretation lacks a formal proof of termination

Deriving Abstract Interpreters from Skeletal Semantics

Thomas Jensen, **Vincent Rébiscoul**, Alan Schmitt

September 18, 2023

INRIA, Université de Rennes

Maintaining the AI-state

- The AI-state is modified when calling a **specified function** (like `eval_stmt`)
- Because the AI-state is language dependent, the modifications must be specified

If

$$\mathcal{A}, \texttt{eval\_stmt}\,(s^\sharp, t^l) \Downarrow^\sharp s'^\sharp, \mathcal{A}'$$

Then

$$\mathcal{A}'(l, \textsc{In}) = \mathcal{A} \sqcup_{\texttt{store}} s^\sharp$$
$$\mathcal{A}'(l, \textsc{Out}) = \mathcal{A} \sqcup_{\texttt{store}} s'^\sharp$$