# Abstract interpreters: A monadic approach to modular verification

Sébastien MICHELLAND<sup>1</sup>, <u>Yannick ZAKOWSKI</u><sup>2</sup>, Laure GONNORD<sup>1</sup> <sup>1</sup> Université Grenoble-Alpes, Grenoble INP, LCIS (Valence) <sup>2</sup> Inria Lyon

November 25th, 2024





# Pitching an internship...

Language description

#### 🔁 REUSABLE

# Pitching an internship...

Language description

#### 🔁 REUSABLE

(Semantic components)<sup>#</sup> state, exceptions...

C

↓ ♂ REUSABLE (Control flow combinators)<sup>#</sup>

Abstract interpreter

# Pitching an internship...



This paper

Abstract interpreters

\_ayered monadic interpreters

Combining both concepts

Free proofs

Conclusion



# Contributions in this paper

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 1/27

# Abstract interpreters: A monadic approach to modular verification ICFP'24 • <u>HAL</u> • <u>ACM</u> • <u>Source code</u>

- 1. Abstract interpreters in layered monadic style
  - IMP and ASM
  - Key idea: proper understanding of control flow
  - Analyzer defined by mirroring interpreter
- 2. Proof of soundness is now modular in terms of language features
  - Meta-theorems for composing components' soundness proofs
  - Components reusable across languages

Alastad bilegedata a Kanado Ayyunah ke	Metabas V.A. Morrell, D. M. Terrat, 2019, New Year (Sec.) Terrat, 2019, New Year (Sec.) Terrat, 2019, New Year (Sec.) Terrat, 2019, New Year (Sec.)	Abstract Interpretors: a Manufic Approach to Modular Varification MUNYHULS ADDODD
Concert program	<ul> <li>Personal assession, Name and Series, Name an</li></ul>	and risk the distance in the set of the level of the distance is the distance in the distance is the distance
I DOWN LAWRY & MARK V A	$\label{eq:result} \begin{array}{ c c c c c c c c c c c c c c c c c c c$	1 STRUMENTAL TO this of the distribution of the distribution of the distribution of the first solid with distribution of the distribution (The distribution of the distrument of the distrument of the dist
- Auf () two (out * a () () () () () () () () () () () () () (	Participation of the second se	4 is usered - order location for one has a beaux was provide the result information of the second results of the second results and the second results are second results and the second results are second rescond rescond
Concess Ffr	3 A TASTE OF GAR LH Before getting tolo tarf using our blowy to simul orde for this given in Figs representation. Thering previously define evolution of their bandled	where applieds, steps horness intramacily instead when its main data for high me is made. In matter, it is a properties approach to a manage approaching their steps. The mean of hermitian approaching the steps. The steps are steps and the step approaching the steps of the steps and the step approaching the steps of the steps and the step approaching the steps of the step approaching the step
E Fig. 6. On Josef from 14 industries 3	withmostic bundless are plant their abstract implementation differed to the constraint of the second	- Stylend January and Annual A Annual Annual Ann

This paper

Abstract interpreters

Layered monadic interpreters

Combining both concepts

Free proofs

Conclusion



# Abstract interpreters

a practical recipe

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 2/2

### A naive analyzer

How to know possible values of variables at runtime?

Run the program!

**One output:** (x, y, z) = (4, 8, 6)

Ok, but... number of inputs? termination? X

## AI (1/2): from collecting semantics to lattices

Let's collect all values anyway.

All outputs:

 $(x,y,z) \in \{(1,11,33),(2,10,15),(3,9,9),(4,8,6),(5,7,3)\}$ 

Excessive amount of values! X

Conclusior

## AI (1/2): from collecting semantics to lattices

Let's collect all values and split the variables.

Upper bound on possible outputs:

 $x \in \{0,1,2,3,4,5\}, \ y \in \{7,8,9,10,11,12\}, \ z \in \{3,6,9,12,15,...,33,36\}$ 

Still too many values! X

Approximate, but still safe.

## AI (1/2): from collecting semantics to lattices

▶ Let's collect all values and split the variables and approximate sets with intervals.

Even upper bound on possible outputs:  $x \in [\![0, 5]\!], y \in [\![7, 12]\!], z \in [\![3, 36]\!]$ 

Tractable

Approximate, but still safe.

# AI (2/2): handling control flow

Control flow depends on values so we might take multiple paths.

$x \leftarrow input \% 6;$	$ ightarrow  x \in \{0,1,2,3,4,5\}$
if $x < 3$	true for $x = 0, 1, 2$
$y \leftarrow x;$	$ ightarrow$ y $\in$ {0, 1, 2}
else	true for $x = 3, 4, 5$
$y \leftarrow 12 - x;$	$ ightarrow \ \mathbf{y} \in \{7, 8, 9\}$
end	$ ightarrow \ y \in \{0,1,2\} \ igcup \ \{7,8,9\}$

Bound on possible outputs:  $x \in \{0, 1, 2, 3, 4, 5\}, y \in \{0, 1, 2, 7, 8, 9\}$ 

► Join paths with a set union.

# AI (2/2): handling control flow

Control flow depends on values so we use algorithms that account for all paths.

$x \leftarrow input \% 6;$	$ ightarrow  {f x} \in \llbracket {f 0}, {f 5}  rbracket$
if $x < 3$	true for $x \in [\![0,2]\!]$
$y \leftarrow x;$	$ ightarrow \ \mathbf{y} \in \llbracket 0, 2  rbracket$
else	true for $x \in \llbracket 3, 5 \rrbracket$
$y \leftarrow 12 - x;$	$\rightarrow y \in \llbracket 12, 12 \rrbracket - \llbracket 3, 5 \rrbracket = \llbracket 7, 9 \rrbracket$
end	$\rightarrow y \in \llbracket \emptyset, 2 \rrbracket \ { { \hspace{65em} [\hspace{.05em} 0, 2]\hspace{05em}]}} = \llbracket \emptyset, 9 \rrbracket$

Bound on possible outputs:  $x \in [\![0,5]\!], \ y \in [\![0,9]\!]$ 

► Join paths with the approximation of a set union.

### Abstract interpreters: recipe

Interpret "normally" but replace as follows:

	Concrete	Abstract
Values	12:int	<pre>[12,12] : interval</pre>
Operators	a + b	$[\![a_1,a_2]\!]+^{\#}[\![b_1,b_2]\!]=[\![a_1+b_1,a_2+b_2]\!]$
Conditions	if e $\{c_1\}$ else $\{c_2\}$ end	Approximate union of values in $c_1 \mbox{ and } c_2$
Loops	while e {c}	Approximate fixpoint of c

- 1. Replace data types with subset approximations (lattices).
- 2. Replace control flow structures with specialized algorithms that account for all paths.

-

This paper

Abstract interpreters

Layered monadic interpreters

Combining both concepts

Free proofs

Conclusion



# Layered monadic interpreters

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 6/27

Conclusior

#### Shallow vs. Deep

(Arguably) more traditional approach:

- Deeply embedded configurations  $\Sigma$  as an inductive
- $\blacktriangleright$  Specify its semantics  $\Sigma \to \Sigma \to \mathbb{P}$

#### Shallow vs. Deep

(Arguably) more traditional approach:

- Deeply embedded configurations Σ as an inductive
- $\blacktriangleright$  Specify its semantics  $\Sigma \to \Sigma \to \mathbb{P}$

What we consider here:

- $\blacktriangleright$  Deeply embedded configurations  $\Sigma$  as an inductive
- $\blacktriangleright$  Shallow representation of those a monadic interpreter:  $[\![\cdot]\!]:\Sigma\to M$

#### Shallow vs. Deep

(Arguably) more traditional approach:

- $\blacktriangleright$  Deeply embedded configurations  $\Sigma$  as an inductive
- $\blacktriangleright$  Specify its semantics  $\Sigma \to \Sigma \to \mathbb{P}$

What we consider here:

- Deeply embedded configurations Σ as an inductive
- Shallow representation of those a monadic interpreter:  $\llbracket \cdot \rrbracket : \Sigma \to M$ Potential benefits:
  - ▶ If *M* is gentle, *may* be executable;
  - ▶ [[·]] *may* be built out of reusable components;
  - $\llbracket \cdot \rrbracket$  may be build structurally over  $\Sigma$ .

#### Monads as models, monads as a programming abstraction

Monad M (for us): a family of types representing a class of effectful programs.

 $\blacktriangleright$  *M R* is the type of programs returning an *R*.

### Monads as models, monads as a programming abstraction

Monad M (for us): a family of types representing a class of effectful programs.

• M R is the type of programs returning an R.

#### Constructors

▶ bind 
$$(p: M T)$$
  $(k: T \rightarrow M R) : M R$ 

And monad-specific operations.

Pure computation Sequence

Pure computation

Seauence

## Monads as models, monads as a programming abstraction

Monad M (for us): a family of types representing a class of effectful programs.

• M R is the type of programs returning an R.

#### Constructors

- ▶ bind (p: M T)  $(k: T \rightarrow M R): M R$
- And monad-specific operations.

Famously central to Haskell, but can also be used in the Coq language (Gallina).

- ret (fibonacci n / 4) : M nat
- ▶ bind p (fun  $x \Rightarrow$  ret (x + 1)) : M nat (assuming p : M nat)

Pure computation

Seauence

## Monads as models, monads as a programming abstraction

Monad M (for us): a family of types representing a class of effectful programs.

• M R is the type of programs returning an R.

#### Constructors

- ▶ bind (p: M T)  $(k: T \rightarrow M R): M R$
- ► And monad-specific operations.

Famously central to Haskell, but can also be used in the Coq language (Gallina).

- ret (fibonacci n / 4) : M nat
- ▶ bind p (fun  $x \Rightarrow ret (x + 1)$ ) : M nat (assuming p : M nat)

And relators, and equations...

## A lightweight extension: monad transformer

**State monad transformer** for state *S* adds to a given monad *M*:

- ▶ get : (stateT M) S
- ▶ set (s:S): (stateT M) unit

Failure monad transformer adds:

▶ abort:(failT M)Ø

Example (executable inside of Coq).

▶ if x = 0 then abort else set (100/x) : failT (stateT M) unit

failT (stateT M) is (almost) fine for IMP, but other languages have different features.
How can theorems talk about "any monad stack"?

#### The freer monad

#### **Freer monad** for events (E : Type $\rightarrow$ Type) has ret, bind and:

```
▶ trigger (e: E T): freerM E T
```

(Not executable)

#### The freer monad

**Freer monad** for events (E : Type  $\rightarrow$  Type) has ret, bind and:

```
► trigger (e: E T): freerM E T
```

(Not executable)

E is a description of the language's operations' signatures.

- Variant stateE := Get : stateE S | Set (s : S) : stateE unit
- ► Variant failE := Abort : failE Ø
  - ▶ freerM stateE  $\approx$  stateT *Id*
  - ▶ freerM (failE + stateE) ≈ failT (stateT Id)

freerM E doesn't implement the events, but it's useful as an intermediate representation.

#### The freer monad

**Freer monad** for events (E : Type  $\rightarrow$  Type) has ret, bind and:

```
► trigger (e: E T): freerM E T
```

(Not executable)

E is a description of the language's operations' signatures.

- Variant stateE := Get : stateE S | Set (s : S) : stateE unit
- ► Variant failE := Abort : failE Ø
  - ▶ freerM stateE  $\approx$  stateT *Id*
  - ▶ freerM (failE + stateE) ≈ failT (stateT Id)

freerM E doesn't implement the events, but it's useful as an intermediate representation.

Interaction Trees [XZHH+20]: (itree E) is (freerM E) with non-termination

#### Algebraic effects and their handlers?

```
Freer monad for events (E : Type \rightarrow Type) has ret, bind and:

trigger (e : E T) : freerM E T
```

(Not executable)

Operations have signatures, but little semantics. What can we do?

Extend the signature to a theory: we get algebraic effects;

- ► Look for their *handlers*.
  - Provide a handler :  $E \rightsquigarrow M$ ;
  - Double check you built a model of your algebra;
  - Get a lifting to computations :  $FreerE \rightsquigarrow M$ .

If you are happy with one shot continuations, implementing this in Coq is easy.

### A layered interpreter for IMP

IMP program: if x = 0 { abort() } else { x = 100/x }



: itree (failE + stateE)

► Control flow structures: sequence (drawn o) and if change signature when handled.

#### A layered interpreter for IMP

IMP program: if x = 0 { abort() } else { x = 100/x }



► Control flow structures: sequence (drawn ○) and if change signature when handled.

#### A layered interpreter for IMP

IMP program: if x = 0 { abort() } else { x = 100/x }



► Control flow structures: sequence (drawn  $\circ$ ) and if change signature when handled.

Combining both concepts

Free proofs

Conclusion

## Where I would like to get to



A semantics for (sequential) LLVM IR built as a layered interpreter using itrees.

- Jourdan et al. analyse C (Verasco),
- Bodin et al. analyse Javascript,
- ► We would like to analyze LLVM IR? For now, we tackle IMP and ASM, but exploring a new methodology.



Conclusior

# Abstract interpreters: A monadic approach to modular verification ICFP'24 • <u>HAL</u> • <u>ACM</u> • <u>Source code</u>

- 1. Abstract interpreters in layered monadic style
  - IMP and ASM
  - Key idea: proper understanding of control flow
  - Analyzer defined by mirroring interpreter
- 2. Proof of soundness is now modular in terms of language features
  - Meta-theorems for composing components' soundness proofs
  - Components reusable across languages

Address temperature to descende temperature to antiprotection Constant temperature to Constant temperature to Constan	The second secon	Alterat Largence: Marcine Control and Con
international and	Frequent [1], do temb te Ro motor with 1 do D or relationships 1 do	and the state of t
- Gald Gran Said + 10 - Convert dia - Convert di - Convert di - Convert dia - Convert dia	Belander (and compared to 0)     Belander (and a solution field     Denoted, 4.01, 5.01, 6     Denoted, 4.01, 5.01, 6     Denoted, 4.01, 5.01, 6     Denoted, 4.01, 7.01, 6	In teaching sharph of a days, and a start with start product to good the start of the start days and the start of the sta
tail (stat) - tail (stat) and jamentable Fig. Co	Below going into teel using our blowy to simul orde for the given in Fig- representation. Thesing processeds defi- evator and their bandler arithmetic handlers are produced about a topic implementation different	with adjoint according to insure any order difficult to the displace of the di
Reads from top to home on S	101.136.1.50b.2006	- Mill Lifes L Adaptive Publication and a con-

This paper

Abstract interpreters

\_ayered monadic interpreters

Combining both concepts

Free proofs

Conclusion

# 4

# Layered monadic abstract interpreters

#### there's got to be a better name

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 14/22

#### On the nature of the "abstract semantics" we build



#### 👃 Very important

The abstract semantics is a hybrid of  $\underline{both}$  the analyzed program and analyzer.

Like an abstract interpreter partially evaluated on a given input program.

#### Can we even build abstract programs with the layered event handling process?

# Hybrid flow impacts event handling

The abstract semantics is a hybrid of  $\underline{both}$  the analyzed program and analyzer.

Handling events with failT adds the ability to crash. But:

A potentially-crashing A tool that analyses tool that analyses is not potentially-crashing pure programs programs

We need "monad transformers" that extend the analysis, not the analyzer.

# Hybrid flow impacts event handling

The abstract semantics is a hybrid of **<u>both</u>** the analyzed program and analyzer.

Handling events with failT adds the ability to crash. But:

A potentially-crashing		A tool that analyses
tool that analyses	<u>is not</u>	potentially-crashing
pure programs		programs

We need "monad transformers" that extend the analysis, not the analyzer. So:

- 1. Implement control flow analyses that know about states/crashes
- 2. Enable these features during event handling

# The key: parameterized control flow algorithms

#### Example: parametrized sequence.

#### Can handle pure programs

may\_exit always false, step always OK

#### Can handle programs in stateT

 $\blacktriangleright T_n = S \times ..., U_n = S \times ...$ 

#### Can handle programs in failT

▶ U<sub>n</sub> = option..., use step/may\_exit

#### Event handling in abstract program:

- 1. Replace events as usual
- 2. Update control flow algorithms' parameters to add state/failure/etc



### The lens that clears it up: monad of control flow

We are in fact describing a freer monad with explicit control flow operations.

Monad of control flow <u>aflow</u> for events (E : Type  $\rightarrow$  Type) has ret, bind and:

▶ trigger (e : E R) Freer monad ▶ seq (p : aflow E T) (k : T → aflow E R) ⟨params...⟩ Source sequence ▶ if (p<sub>1</sub> p<sub>2</sub> : aflow E R) ⟨params...⟩ Source conditional ▶ ... do, while, cfg...

New notion of event handling:

- 1. Replace events like before
- 2. Also update parameters of control flow analysis algorithms to enable state/failure

#### And now: a monadic abstract program

Need abstract events because their parameters/return values become lattices.



#### This time the if changes a lot with each handling.

#### Deriving both programs from a single denotation



#### Shared SurfaceAST representation:

- Control flow combinator tree (later projected to itree and aflow)
- Leaves are ret or trigger with pairs: (2, [[2, 2]]), (Get, Get<sup>#</sup>)

#### Technical aside: combinators, a closer look

We keep aflow E R fairly minimal:

- ▶ **Ret** (*x* : *R*)
- ▶ Trigger (e : E R)
- ▶ Seq ( $f_1$  : aflow E  $U_1$ ) ( $f_2$  :  $T_1$  → aflow E R) (step :  $U_1$  →  $T_1$ ) (may\_exit :  $U_1$  → bool) (merge : bool →  $U_1$  → R → R)
- ► Fixpoint (...)
- ► TailMrec (...)

Higher level combinators (if, do, cfg,..) are analyzes implemented directly in aflow. A: They still must specify how state and fail update their parameters!

## Our implementation

Our Coq development: https://gitlab.inria.fr/sebmiche/itree-ai

Everything formalized and packaged in a library.

- Monad theory, aflow, shared denotations with SurfaceAST
- Basic lattices and non-relational domains
- Control flow: seq, if, do, while, cfg

Enough to write two case studies, i.e., abstract interpreters for:

- IMP with arithmetic, state, and failure handled as three successive layers;
- ► ASM with two layers of state (registers and heap).

 $\rightsquigarrow$  And of course, executable through extraction.

### Our implementation

Our Coq development: https://gitlab.inria.fr/sebmiche/itree-ai

Everything formalized and packaged in a library.

- Monad theory, aflow, shared denotations with SurfaceAST
- Basic lattices and non-relational domains
- Control flow: seq, if, do, while, cfg

Enough to write two case studies, i.e., abstract interpreters for:

- IMP with arithmetic, state, and failure handled as three successive layers;
- ► ASM with two layers of state (registers and heap).
- $\rightsquigarrow$  And of course, executable through extraction.

But are those analyzer sound?

This pape

Abstract interpreters

\_ayered monadic interpreters

Combining both concepts

Free proofs

Conclusion



# All the proofs are now much easier

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 22/27

# Trying it out: proving an IMP analyzer

Syntax and semantics	USER
Using concrete/abstract control flow pairs from library	
Numerical domain: $\mathbb{Z}$ interval lattice	REUSABLE
Soundness of layer #1 (failT): assertion	assert()
IMP-specific because involves truth values	USER
Soundness of layer #2 (stateT): variables	$x, y, z \leftarrow$
▶ Basically just a map lattice string $\rightarrow$ value	REUSABLE
Soundness of flow analysis algorithms	Meta-theory
Composing layers' soundness proofs	LIBRARY

Combining both concepts

Conclusion

### Bird's eye view



The sound' invariant maintains that:

the control flow structure of both computations match;

matching events and values are related through Galois connections. If programmed through the DSL, for free.



Abstract interpreters

Layered monadic interpreters

Combining both concepts

Free proofs ○●○ Conclusior



We need to preserve sound' by failure interpretation:

- the user defined handlers must be proven sound;
- the library does the rest.



We keep going...

Cambium Seminar (Inria, Paris)

ayered monadic interpreters

Combining both concepts

Conclusior

## Bird's eye view



Finally, we unfold the implementation of the abstract algorithms: each pair is proven relatively sound in the library, allowing us to conclude.

Combining both concepts

## Certified analysis checklist

Obligation	Who	When
Mirroring of concrete/abstract denotations	User	Every language
Lattice and domains (intervals)	User	Only once
Language features (state, failure)	User	Only once
Soundness of parametrized flow algorithms	Library	Every flow structure
Soundness of event handling steps	Library	Every flow structure
Composition of event handling steps	Library	Only one

Analyses for languages features are thus:

- Modular (proven independently then composed)
- Reusable (break, local variables, abort()... often the same)

This pape

Abstract interpreters

\_ayered monadic interpreters

Combining both concepts

Free proofs

Conclusion



# Conclusion

Cambium Seminar (Inria, Paris)

Abstract interpreters: a monadic approach to modular verification 25/27

### More details in the paper, and even more in the code!

The paper: https://hal.science/hal-04628727 (ICFP'24) **The code:** https://gitlab.inria.fr/sebmiche/itree-ai





#### Abstract Interpreters: a Monadic Approach to Modular Verification

ANONYMOUS AUTHOR(S)

We argue that monadic interpreters built as layers of handlers stacked atop the free monad, as advocated ve argue nan menanza anna peers anna an ay ca sa nataanas naenen atoy na nee monaa, ao aoronaecu notably by the ITree library, alao constitute a promising way to implement and verify abstract interpreters in totany by the stree nurary, not construine a promising way to imprementation of dependently-typed theories such as the one underlying the Coq proof assistant. spennessny-typeu useones and as tae one uncertying toe coq proor asastant. The approach enables both code reuse across projects and modular proofs of soundness of the resulting

a ne approach evantes nonn coor reture across projects anu monutar proots or southaness or the returning interpreters. We provide generic abstract control flow combinators proven correct once and for all against their concrete counterpart. We demonstrate how to relate concrete handlers implementing effects to abstract uses concrete sounderpart, we sensitize the reading of the reading sound reading sources to instruct variants of these handlers, essentially capturing the traditional soundness of transfer functions in the context variants or mese namers, essentiatoy capturing for transmona sourness or themest nuceation in our volues, of monade interpreters. Finally, we provide generic results to lift soundness statements via the interpretation

) a neuron anno namore enex or. We formalize all the aforementioned combinators and theories into a Coq library, and demonstrate their

we commance on one nonemenous commonstration and incomes more a contrast, which endotronistic time benefits by implementing and proving correct two illustrative abstract interpreters respectively for a structured

1 INTRODUCTION

Monad-based abstract interpreters are modular and their proofs are too!

#### Novelties

- ► Abstract interpreters in layered monadic style + soundness tools
- Identifying the freer monad of control flow

#### Insights and future work

- Scaling up: new effects; less structured control flow; better analysis algorithms; combining domains...
- Performances: at the moment, unfold into itrees, then extract.

Monad-based abstract interpreters are modular and their proofs are too!

#### Novelties

- ▶ Abstract interpreters in layered monadic style + soundness tools
- Identifying the freer monad of control flow

#### Insights and future work

- Scaling up: new effects; less structured control flow; better analysis algorithms; combining domains...
- Performances: at the moment, unfold into itrees, then extract.

#### Thoughts?