Introduction to abstract interpretation
Part I

Mehdi Naima[1]

May 17, 2021

Université Sorbonne Paris-Nord, LIPN.

Table of contents

Introduction

Points du jour

- Welcome new young people in the lab.

Points du jour

- Welcome new young people in the lab.
- New responsible for the seminar.

Points du jour

- Welcome new young people in the lab.
- New responsible for the seminar.
- Suggestion for next year.

Points du jour

- Welcome new young people in the lab.
- New responsible for the seminar.
- Suggestion for next year.
- Upcoming seminars: part II of this talk and Dina's talk.

Material

- My slides are based mostly on Antoine Miné's lecture notes (freely available on his webpage) and
- two video courses link and link.

Introduction

"Testing is not sufficient!"

Image for illustration purposes only

Introduction

"Testing is not sufficient!"

- There are many famous documented bugs even in well tested softwares.

Image for illustration purposes only

Introduction

"Testing is not sufficient!"

- There are many famous documented bugs even in well tested softwares.
- We should use formal methods to provide rigorous, mathematical insurance of correctness even though we can not prove everything.

Image for illustration purposes only

Introduction

"Testing is not sufficient!"

- There are many famous documented bugs even in well tested softwares.
- We should use formal methods to provide rigorous, mathematical insurance of correctness even though we can not prove everything.
- Correctness properties are undecidable! An automatic general method is impossible to find (we have to find compromises).

Image for illustration purposes only

Formal methods

Formal methods

From [BK08] " To put it in a nutshell, formal methods can be considered as "the applied mathematics for modeling and analyzing ICT systems"

Formal methods

From [BK08] " To put it in a nutshell, formal methods can be considered as "the applied mathematics for modeling and analyzing ICT systems"

Dijkstra "testing can only show the presence of errors, not their absence."

Formal methods

From [BK08] “ To put it in a nutshell, formal methods can be considered as “the applied mathematics for modeling and analyzing ICT systems”

Dijkstra “testing can only show the presence of errors, not their absence.”

Cousot and Cousot in [CC10] classify current formal methods into three categories

- Deductive methods.
- Model Checking.
- Static Analysis.

Abstract interpretation is a theory of approximation and analysis of program semantics that belongs to the category of *Static Analysis*.

Deductive Methods

- Pioneered by Hoare [Hoa69] and Floyd [Flo93].

Deductive Methods

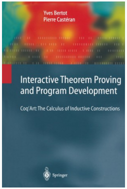
- Pioneered by Hoare [Hoa69] and Floyd [Flo93].
- Relies mostly on the user.

Deductive Methods

- Pioneered by Hoare [Hoa69] and Floyd [Flo93].
- Relies mostly on the user.
- A system attempts to produce a formal proof, given a description of the system, with a set of logical axioms and inference rules.

Deductive Methods

- Pioneered by Hoare [Hoa69] and Floyd [Flo93].
- Relies mostly on the user.
- A system attempts to produce a formal proof, given a description of the system, with a set of logical axioms and inference rules.
- Examples include proof assistants such as Coq developed by Bertot and Castéran in [BC13].



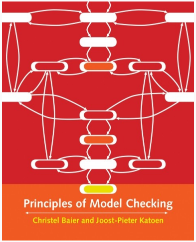
Model Checking

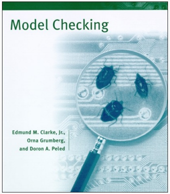
From [BK08] "Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model."

Model Checking

From [BK08] “Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.”

- Pioneered by Clarke, Emerson, Sifakis and Queille.

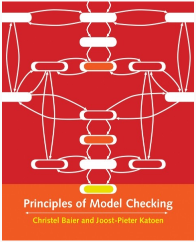


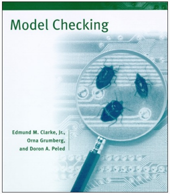


Model Checking

From [BK08] "Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model."

- Pioneered by Clarke, Emerson, Sifakis and Queille.
- Book [BK08, CJGK^{+}18].

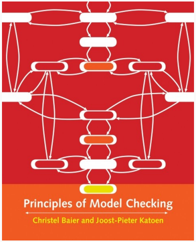


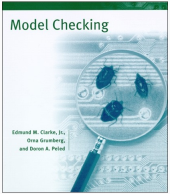


Model Checking

From [BK08] "Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model."

- Pioneered by Clarke, Emerson, Sifakis and Queille.
- Book [BK08, CJGK+18].
- System verifies certain properties by means of an exhaustive search.

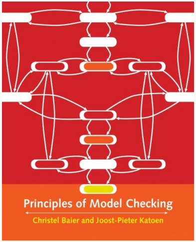


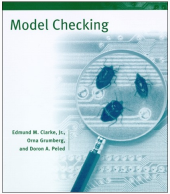


Model Checking

From [BK08] "Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model."

- Pioneered by Clarke, Emerson, Sifakis and Queille.
- Book [BK08, CJGK+18].
- System verifies certain properties by means of an exhaustive search.
- Checks the absence of errors (i.e., property violations) and alternatively can be considered as an intelligent and effective debugging technique.

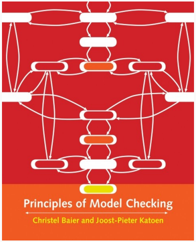


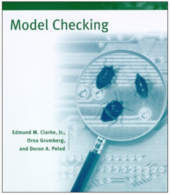


Model Checking

From [BK08] "Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model."

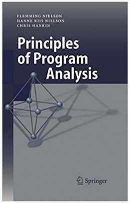
- Pioneered by Clarke, Emerson, Sifakis and Queille.
- Book [BK08, CJGK+18].
- System verifies certain properties by means of an exhaustive search.
- Checks the absence of errors (i.e., property violations) and alternatively can be considered as an intelligent and effective debugging technique.
- Software Examples include CBMC.





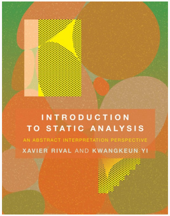
Static Analysis

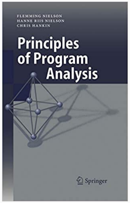
- system verifies an over-approximation of a behavioural property of the program.



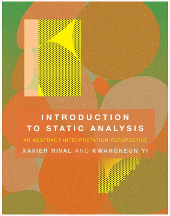
Static Analysis

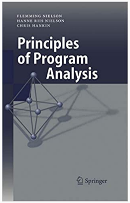
- system verifies an over-approximation of a behavioural property of the program.
- References include [NNH04, RY20].





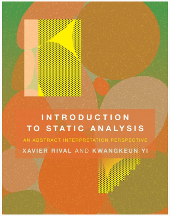
- system verifies an over-approximation of a behavioural property of the program.
- References include [NNH04, RY20].
- Works directly on the source code and infer properties on program executions automatically.

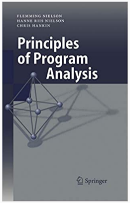




Static Analysis

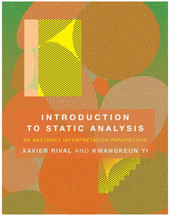
- system verifies an over-approximation of a behavioural property of the program.
- References include [NNH04, RY20].
- Works directly on the source code and infer properties on program executions automatically.
- Signals all possible Run Time Errors (soundness). This includes division by zero, bounds array indexing, integers and arithmetic overflow. . .

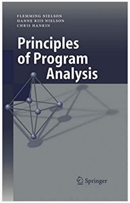




Static Analysis

- system verifies an over-approximation of a behavioural property of the program.
- References include [NNH04, RY20].
- Works directly on the source code and infer properties on program executions automatically.
- Signals all possible Run Time Errors (soundness). This includes division by zero, bounds array indexing, integers and arithmetic overflow. . .
- It also signal some errors that cannot really happen (false alarms on spurious executions e.g. when hypotheses on the execution environment are not taken into account).





Abstract Interpretation

- Part of static analysis methods.

Abstract Interpretation

- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]

Abstract Interpretation

- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.

Abstract Interpretation

- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.

Abstract Interpretation

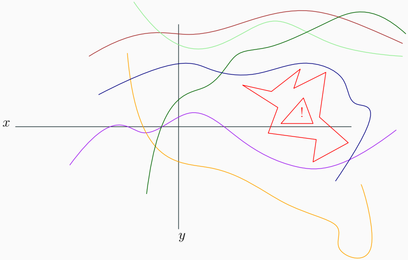
- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.
- Guides design to ensure soundness by construction.

Abstract Interpretation

- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.
- Guides design to ensure soundness by construction.
- Examples of softwares include Astrée

Abstract Interpretation

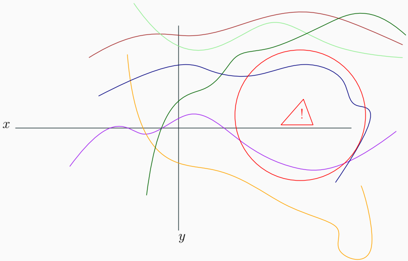
- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.
- Guides design to ensure soundness by construction.
- Examples of softwares include Astrée



Concrete domains of program properties are hard to follow, create tailored abstract domains that overapproximate the concrete ones.

Abstract Interpretation

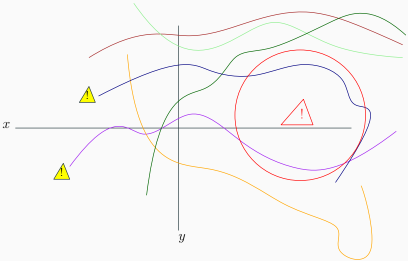
- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.
- Guides design to ensure soundness by construction.
- Examples of softwares include Astrée



Concrete domains of program properties are hard to follow, create tailored abstract domains that overapproximate the concrete ones.

Abstract Interpretation

- Part of static analysis methods.
- Introduced by Cousot and Cousot in [CC77]
- a general theory of the approximation of formal program semantics based on monotonic functions over ordered sets, especially lattices.
- It unifies existing semantics.
- Guides design to ensure soundness by construction.
- Examples of softwares include Astrée



Concrete domains of program properties are hard to follow, create tailored abstract domains that overapproximate the concrete ones.

Order Theory

Partial orders

Definition (Partial Order, Poset)

A partial order $\sqsubseteq$ on a set X is a binary relation over X such that:

- reflexive: $\forall x \in X, x \sqsubseteq x$;
- anti-symmetric: $\forall x, y \in X, (x \sqsubseteq y) \wedge (y \sqsubseteq x) \implies (x = y)$;
- transitive: $\forall x, y, z \in X, (x \sqsubseteq y) \wedge (y \sqsubseteq z) \implies (x \sqsubseteq z)$.

Partial orders

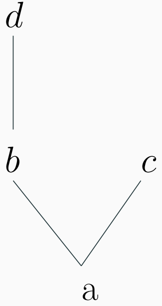
Definition (Partial Order, Poset)

A partial order $\sqsubseteq$ on a set X is a binary relation over X such that:

- reflexive: $\forall x \in X, x \sqsubseteq x$;
- anti-symmetric: $\forall x, y \in X, (x \sqsubseteq y) \wedge (y \sqsubseteq x) \implies (x = y)$;
- transitive: $\forall x, y, z \in X, (x \sqsubseteq y) \wedge (y \sqsubseteq z) \implies (x \sqsubseteq z)$.

Let $X = \{a, b, c, d\}$, then $R = \{(a, a), (b, b), (c, c), (d, d)(a, b), (a, c), (b, d), (a, d)\}$ is a partial order.

Representation with Hasse Diagram.



Poset examples

$\dots$

3

2

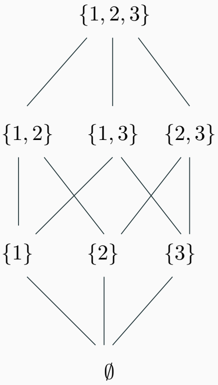
1

0

Integer poset (totally ordered)

Poset examples

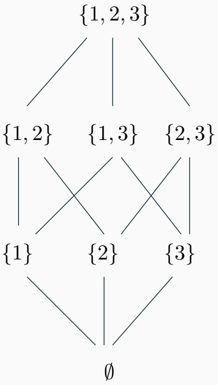
Integer poset (totally ordered)



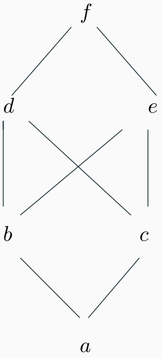
Powerset Poset

Poset examples

Integer poset (totally ordered)



Powerset Poset



Poset

Lattices

- Given two elements a and b in a poset X, an **upper bound** is any element $c \in X$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$.
- Moreover, c is the least upper bound or **join** "$\sqcup$", if it exists is the smallest element greater than both a and b.
- These notions have their symmetric the **lower bound** and greatest lower bound (or **meet** "$\sqcap$").

Lattices

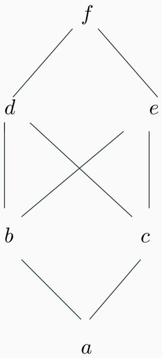
- Given two elements a and b in a poset X, an **upper bound** is any element $c \in X$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$.
- Moreover, c is the least upper bound or **join** "$\sqcup$", if it exists is the smallest element greater than both a and b.
- These notions have their symmetric the **lower bound** and greatest lower bound (or **meet** "$\sqcap$").

Definition (Lattice)

Lattices are particular posets which contain more structure. They are posets where the meet and join always exist for pairs of elements.

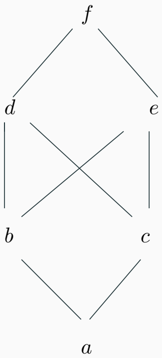
If the property holds for arbitrary set the lattice is said to be **complete**.

Lattice examples

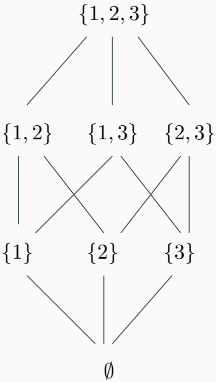


Not a lattice, *d* and *e* do not have a meet.

Lattice examples

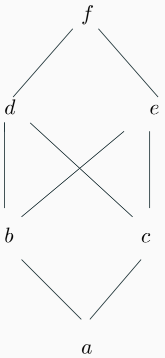


Not a lattice, *d* and *e* do not have a meet.

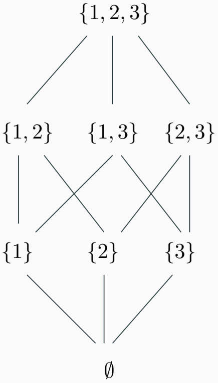


Powerset poset is a complete lattice

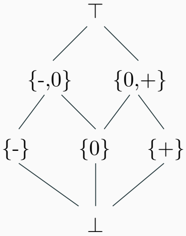
Lattice examples



Not a lattice, *d* and *e* do not have a meet.



Powerset poset is a complete lattice



Sign poset is a complete lattice

Fixpoints

Fixpoints

Definition

Given a poset $(X, \sqsubseteq)$ and an operator $f : X \to X$:

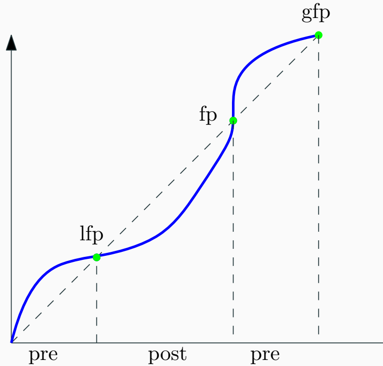
- x is a **fixpoint** of f if $x = f(x)$. We denote as $fp(f) = \{x \in X | f(x) = x\}$ the set of fixpoints of f.
- x is a **prefixpoint** of f if $x \sqsubseteq f(x)$.
- x is a **postfixpoint** of f if $f(x) \sqsubseteq x$.
- $lfp\, f = \min\{y \in fp(f)\}$ if it exists is called the **least fixpoint** of f.

Fixpoints

Definition

Given a poset $(X, \sqsubseteq)$ and an operator $f : X \to X$:

- x is a **fixpoint** of f if $x = f(x)$. We denote as $fp(f) = \{x \in X | f(x) = x\}$ the set of fixpoints of f.
- x is a **prefixpoint** of f if $x \sqsubseteq f(x)$.
- x is a **postfixpoint** of f if $f(x) \sqsubseteq x$.
- $lfp\, f = \min\{y \in fp(f)\}$ if it exists is called the **least fixpoint** of f.

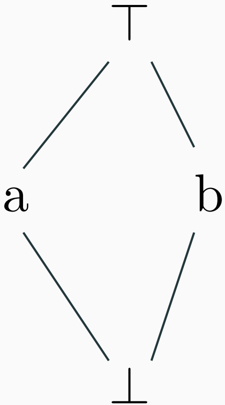


Fixpoint examples

Definition

$$f(\top) = \top$$
$$f(a) = a$$
$$f(b) = \top$$
$$f(\bot) = a$$

$$fp(f) = \{a, \top\}$$
$$lfp(f) = a$$
$$gfp(f) = \top$$
$$pre(f) = \{\bot, b\}$$
$$post(f) = \emptyset$$



Fixpoint examples

Definition

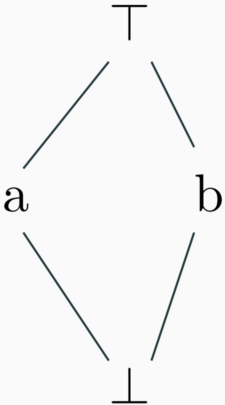
$$f(\top) = \top$$
$$f(a) = a$$
$$f(b) = \top$$
$$f(\bot) = a$$

$$fp(f) = \{a, \top\}$$
$$lfp(f) = a$$
$$gfp(f) = \top$$
$$pre(f) = \{\bot, b\}$$
$$post(f) = \emptyset$$

Definition

$$f(\top) = b$$
$$f(a) = \top$$
$$f(b) = a$$
$$f(\bot) = a$$

$$fp(f) = \emptyset$$
$$pre(f) = \{\bot, a\}$$



Knaster-Tarski Theorem

Definition

A function f in a complete lattice S is **monotone** if

$$\forall, x, y \in S, \quad x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$$

Theorem ([T+55])

Let S be a complete lattice and f a monotone function on S then f admits a least fixpoint

$$\mathit{lfp}\, f = \bigsqcap \{x \in S, f(x) \sqsubseteq x\}.$$

Approximations

Concretization

When we are given a concrete and an abstract domain we need to be able to pass from one to another. This is done with two functions that are called **concretization** and **abstraction**.

Concretization

When we are given a concrete and an abstract domain we need to be able to pass from one to another. This is done with two functions that are called **concretization** and **abstraction**.

Definition

A **concretization** function $\gamma \in (A, \sqsubseteq) \rightarrow (C, \leq)$ is a *monotone function* assigning a concrete meaning, in C to each abstract element in A.

Concretization

When we are given a concrete and an abstract domain we need to be able to pass from one to another. This is done with two functions that are called **concretization** and **abstraction**.

Definition

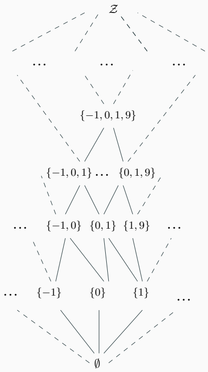
A **concretization** function $\gamma \in (A, \sqsubseteq) \rightarrow (C, \leq)$ is a *monotone function* assigning a concrete meaning, in C to each abstract element in A.

Example

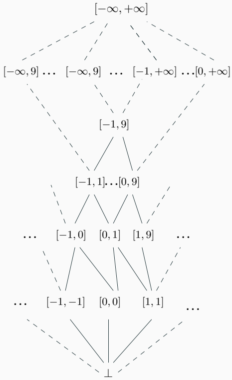
Let the concrete domain be the complete integer powerset lattice and the abstract domain be the interval lattice. Then elements of the abstract domain are intervals of the form $[a, b]$ with $a \geq b$. Then

$$\gamma([a, b]) = \{x \mid a \leq x \leq b\}$$

Concretization

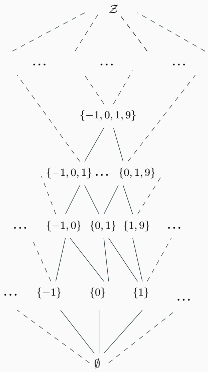


Powerset integer domain $\mathcal{C}$



Interval domain $\mathcal{A}$

Concretization

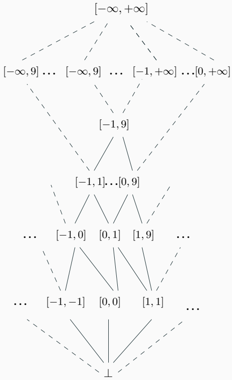


Powerset integer domain $\mathcal{C}$

The concretization function $\gamma : \mathcal{A} \to \mathcal{C}$ is defined by

$$\gamma([a,b]) = \{x \mid a \le x \le b\}$$
$$\gamma(\perp) = \emptyset$$

For example $\gamma([1,4]) = \{1,2,3,4\}$



Interval domain $\mathcal{A}$

Abstraction

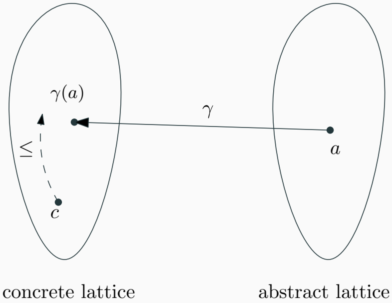
Definition (Sound abstraction)

Let $(A, \sqsubseteq), (C, \leq)$ be two posets. $a \in A$ is a **sound abstraction** of $c \in C$ if and only if $c \leq \gamma(a)$.

Abstraction

Definition (Sound abstraction)

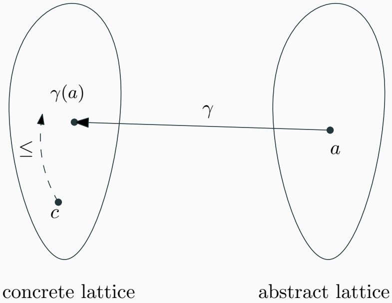
Let $(A, \sqsubseteq), (C, \leq)$ be two posets. $a \in A$ is a **sound abstraction** of $c \in C$ if and only if $c \leq \gamma(a)$.



Abstraction

Definition (Sound abstraction)

Let $(A, \sqsubseteq), (C, \leq)$ be two posets. $a \in A$ is a **sound abstraction** of $c \in C$ if and only if $c \leq \gamma(a)$.



Example

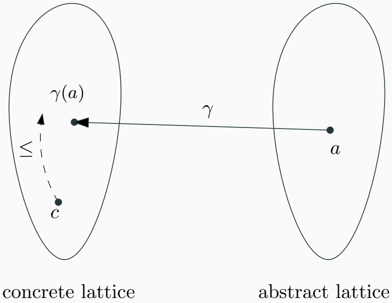
With the same concrete and abstract domains abstraction function $\alpha : \mathcal{C} \to \mathcal{A}$ is defined by

$$\alpha(c) = [\min(c), \max(c)]$$

Abstraction

Definition (Sound abstraction)

Let $(A, \sqsubseteq), (C, \leq)$ be two posets. $a \in A$ is a **sound abstraction** of $c \in C$ if and only if $c \leq \gamma(a)$.



Example

With the same concrete and abstract domains abstraction function $\alpha : \mathcal{C} \to \mathcal{A}$ is defined by

$$\alpha(c) = [\min(c), \max(c)]$$

Therefore $(\alpha \circ \gamma)(\{1,3,5\}) = \{1,2,3,4,5\}$ and $\{1,3,5\} \subseteq \{1,2,3,4,5\}$
We loose information when passing through the abstract domain but the result is sound (we get an overapproximation of the initial value).

Galois connection

- a monotonic concretization γ is sufficient to reason about soundness.

Galois connection

- a monotonic concretization γ is sufficient to reason about soundness.
- more structure, if available, can help us design sound and accurate analyses.

Galois connection

- a monotonic concretization γ is sufficient to reason about soundness.
- more structure, if available, can help us design sound and accurate analyses.
- In the standard Abstract interpretation framework we assume additionally the existence of a monotonic abstraction function $\alpha : C \rightarrow A$ that associates an abstract element to a concrete one such that (α, γ) forms a Galois connection

Galois connection

Definition (Galois connection)

Given two posets $(A, \sqsubseteq)$ and $(C, \leq)$, the pair $(\alpha : C \to A,\ \gamma : A \to C)$ is a Galois connection if:

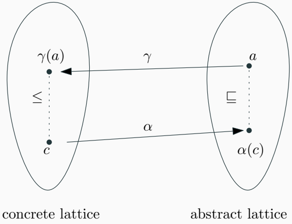
$$\forall a \in A, c \in C, c \leq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a.$$

Galois connection

Definition (Galois connection)

Given two posets $(A, \sqsubseteq)$ and $(C, \leq)$, the pair $(\alpha : C \to A,\ \gamma : A \to C)$ is a Galois connection if:

$$\forall a \in A, c \in C, c \leq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a.$$



Galois connection

Definition (Galois connection)

Given two posets $(A, \sqsubseteq)$ and $(C, \leq)$, the pair $(\alpha : C \to A,\ \gamma : A \to C)$ is a Galois connection if:

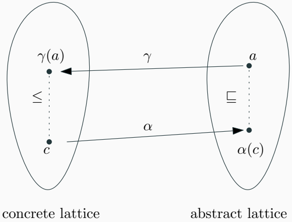
$$\forall a \in A, c \in C, c \leq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a.$$

Properties of Galois connections:

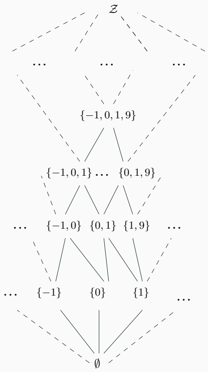
- $\gamma \circ \alpha$ is extensive : $\forall c, c \leq \gamma(\alpha(c))$.
- $\alpha \circ \gamma$ is extensive : $\forall a, \alpha(\gamma(a)) \sqsubseteq a$.
- α and γ are monotonic.

α and γ are called adjoint functions and each one of them can be defined in term of the other.

For instance $\alpha(c) = \sqcap\{a \mid c \leq \gamma(a)\}$



Galois connection examples



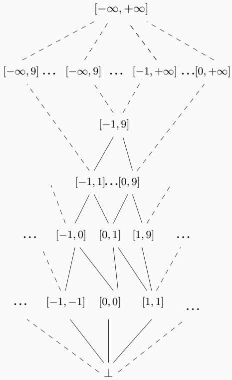
Powerset integer domain $\mathcal{C}$

The concretization function $\gamma : \mathcal{A} \rightarrow \mathcal{C}$ is defined by

$$\gamma([a,b]) = \{x \mid a \leq x \leq b\}$$
$$\gamma(\perp) = \emptyset$$

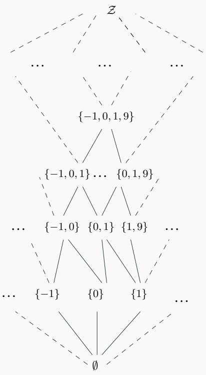
and the abstraction function $\alpha : \mathcal{C} \rightarrow \mathcal{A}$ is defined by

$$\alpha(c) = [\min(c), \max(c)]$$



Interval domain $\mathcal{A}$

Galois connection examples



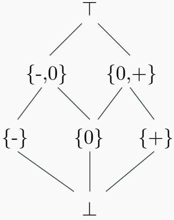
Powerset integer domain $\mathcal{C}$

The concretization function $\gamma : \mathcal{A} \to \mathcal{C}$ is defined by

$$\begin{aligned}
\gamma(\top) &= \mathbb{Z} \\
\gamma(\{-\}) &= \{x \mid x < 0\} \\
\gamma(\{+\}) &= \{x \mid x > 0\} \\
\gamma(\{-,0\}) &= \{x \mid x \leq 0\} \\
\gamma(\{0,+\}) &= \{x \mid x \geq 0\} \\
\gamma(\bot) &= \emptyset
\end{aligned}$$

and the abstraction function $\alpha : \mathcal{C} \to \mathcal{A}$ is defined by

$$\alpha(c) = \begin{cases} \{+\} \text{ if } c \in \mathbb{Z}_{>0} \\ \{-\} \text{ if } c \in \mathbb{Z}_{<0} \\ \{0,+\} \text{ if } c \in \mathbb{Z}_{\geq 0} \\ \{-,0\} \text{ if } c \in \mathbb{Z}_{\leq 0} \\ \bot \text{ if } c = \emptyset \\ \top \text{ else} \end{cases} \tag{1}$$



Abstract sign domain $\mathcal{A}$

Best operator abstraction

Every operation that can happen on the concrete domain has to have an abstract equivalent!

Best operator abstraction

Every operation that can happen on the concrete domain has to have an abstract equivalent!

Definition (Sound operator abstraction)

Given a concretization γ from an abstract domain $(A, \sqsubseteq)$ to a concrete domain $(C, \leq)$, a concrete operator $f : C \rightarrow C$, and an abstract operator $g : A \rightarrow A$:
g is a sound abstraction of f if $\forall a \in A : f(\gamma(a)) \leq \gamma(g(a))$

Best operator abstraction

Every operation that can happen on the concrete domain has to have an abstract equivalent!

Definition (Sound operator abstraction)

Given a concretization γ from an abstract domain $(A, \sqsubseteq)$ to a concrete domain $(C, \leq)$, a concrete operator $f : C \to C$, and an abstract operator $g : A \to A$:
g is a sound abstraction of f if $\forall a \in A : f(\gamma(a)) \leq \gamma(g(a))$

We know that the best abstraction of c is $\alpha(c)$ from the definition ($\alpha(c) = \sqcap\{a \mid c \leq \gamma(a)\}$). From Galois connection, and if g is a sound abstraction

$$c \leq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a$$

$$(f \circ \gamma)(a) \leq (\gamma \circ g)(a) \Leftrightarrow (\alpha \circ f \circ \gamma)(a) \sqsubseteq g(a).$$

Best operator abstraction

Every operation that can happen on the concrete domain has to have an abstract equivalent!

Definition (Sound operator abstraction)

Given a concretization γ from an abstract domain $(A, \sqsubseteq)$ to a concrete domain $(C, \leq)$, a concrete operator $f : C \to C$, and an abstract operator $g : A \to A$:
g is a sound abstraction of f if $\forall a \in A : f(\gamma(a)) \leq \gamma(g(a))$

We know that the best abstraction of c is $\alpha(c)$ from the definition ($\alpha(c) = \sqcap\{a \mid c \leq \gamma(a)\}$). From Galois connection, and if g is a sound abstraction

$$c \leq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a$$

$$(f \circ \gamma)(a) \leq (\gamma \circ g)(a) \Leftrightarrow (\alpha \circ f \circ \gamma)(a) \sqsubseteq g(a).$$

The lattice structure gives an automatic abstract operation

Definition (Best operator abstraction)

Given a Galois connection $(C, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ and a concrete operator $f : C \to C$, the best abstraction of f is given by $\alpha \circ f \circ \gamma$.

Example of best operator abstraction

Definition (Best operator abstraction)

Given a Galois connection $(C, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ and a concrete operator $f : C \to C$, the best abstraction of f is given by $\alpha \circ f \circ \gamma$.

Example of best operator abstraction

Definition (Best operator abstraction)

Given a Galois connection $(C, \leq) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (A, \sqsubseteq)$ and a concrete operator $f : C \to C$, the best abstraction of f is given by $\alpha \circ f \circ \gamma$.

Example

Consider the Galois connection between powerset integer domain and interval domain. Let the operator $f : C \to C$ in the powerset domain be defined as $f(X) = \{x + 1 \mid x \in X\}$. $f(\{1, 3, 4\}) = \{2, 4, 5\}$.
The best abstraction of f in the abstract domain of intervals is thus defined by $f^\sharp = (\alpha \circ f \circ \gamma)$, then for example, $\alpha(\{1, 3, 4\}) = [1, 4]$ and $f^\sharp([1, 4]) = [2, 5]$.

Composition of best operator abstractions is not necessarily the best operator abstraction

That is if g and g' are the best abstractions of f and f' , then $(g \circ g')$ is not always the best abstraction!

Fixpoint transfer

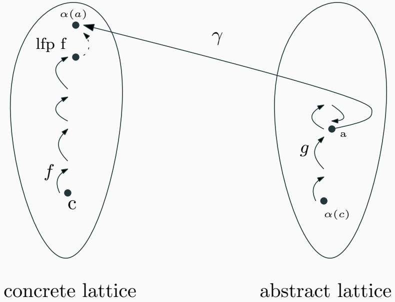
Critical parts of the semantics of a program are defined as least fixpoints $\mathit{lfp}\, f$ of some monotonic or continuous operator $f : C \to C$ in the concrete domain $(C, \leq)$.
In order to abstract $\mathit{lfp}\, f$ in an abstract domain $(A, \sqsubseteq)$, a natural idea is to start with a sound abstraction $g : A \to A$ of f.

Fixpoint transfer

Critical parts of the semantics of a program are defined as least fixpoints *lfp f* of some monotonic or continuous operator $f : C \to C$ in the concrete domain $(C, \leq)$.
In order to abstract *lfp f* in an abstract domain $(A, \sqsubseteq)$, a natural idea is to start with a sound abstraction $g : A \to A$ of f.

Theorem (Tarskian fixpoint transfer)

Given a complete lattice concrete domain, a monotonic concrete function $f : C \to C$, and a sound abstraction $g : A \to A$ in a poset abstract domain, then any postfixpoint a of g is a sound abstraction of lfp f, i.e lfp f $\leq \gamma(a)$



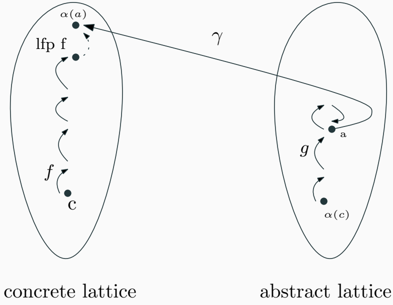
Fixpoint transfer

Critical parts of the semantics of a program are defined as least fixpoints *lfp f* of some monotonic or continuous operator $f : C \to C$ in the concrete domain $(C, \leq)$.
In order to abstract *lfp f* in an abstract domain $(A, \sqsubseteq)$, a natural idea is to start with a sound abstraction $g : A \to A$ of f.

Theorem (Tarskian fixpoint transfer)

Given a complete lattice concrete domain, a monotonic concrete function $f : C \to C$, and a sound abstraction $g : A \to A$ in a poset abstract domain, then any postfixpoint a of g is a sound abstraction of lfp f, i.e lfp f $\leq \gamma(a)$

the theorem can be applied in the useful case where abstract fixpoints are hard to compute, or do not even exist at all. Sometimes fixpoints exist in the concrete, but are not guaranteed to exist in the abstract.



To keep in mind for next time

- Concrete and abstract domains.
- Order theory and lattices.
- Galois connections.
- Fixpoints transfer theorem.

To keep in mind for next time

- Concrete and abstract domains.
- Order theory and lattices.
- Galois connections.
- Fixpoints transfer theorem.

Thank you for your attention

References

[BC13] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

References ii

[CC10] Patrick Cousot and Radhia Cousot. A gentle introduction to formal verification of computer systems by abstract interpretation, 2010.

[CJGK+18] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[Flo93] Robert W Floyd. Assigning meanings to programs. In *Program Verification*, pages 65–81. Springer, 1993.

[Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[NNH04] Flemming Nielson, Hanne R Nielson, and Chris Hankin. *Principles of program analysis*. Springer Science & Business Media, 2004.

[RY20] Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. Mit Press, 2020.

[T+55] Alfred Tarski et al. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.