Abstraction

Material covered in chapter 3.2 of
Introduction to Static Analysis: an Abstract Interpretation Perspective

Purpose of this lecture

Static analysis performs computation over **restricted sets of logical predicates**:

- classical semantics (previous lecture) contains too much information
- not all this information is useful to infer interesting properties
- static analysis needs to rely on an efficient representation of data

This lecture addresses this points by **defining semantic abstraction**:

1. choice of a set of logical predicates
2. definition of the logical tie between these predicates and the actual program properties

Content of the lecture:

- formalization of the notion of abstraction
- presentation of a few standard abstractions

Outline

Ordering over properties

In this slide and the next one, we consider specifically **state properties**, i.e., logical properties overy program states:

State property

A **state property** is a logical predicate over states and it can be defined by the **set of states** that meet this property.

Notation to better distinguish the set view and the predicate view:

$P_{\mathrm{pred}}(s)$ if and only if $s \in P_{\mathrm{set}}$.

Then, for all state properties P, P',

$$\begin{array}{lcl} P \text{ is logically stronger than } P' & \Longleftrightarrow & \forall s,\ P_{\mathrm{pred}}(s) \Longrightarrow P'_{\mathrm{pred}}(s) \\ & \Longleftrightarrow & P_{\mathrm{set}} \subseteq P'_{\mathrm{set}} \end{array}$$

Logical implication is deeply tied to set inclusion

More generally, logical implication defines an ordering

Tying abstract predicates with concrete elements

Definition elements:

- sets of program states, logically ordered by set inclusion
- **abstract states** describing sets of program states and manipulated by the analysis

Examples:

- abstract predicate $\mathtt{x} \in [3, 7]$ describes all states that map $\mathtt{x}$ to a value comprised between 3 and 7
- abstract predicate $\mathtt{x} \geq 0$ describes all states that map x to a non-negative value
- abstract predicate $\mathtt{x} \in [3, 7]$ is stronger than abstract predicate $\mathtt{x} \geq 0$

We should **formalize relations between predicates**:

- logical strength comparison across abstract predicates
- comparison between a concrete set and an abstract predicate

Abstraction relation

Assumption: a concrete domain defined by

- $\mathbb{C}$: set of concrete behaviors
  behaviors may be, e.g., sets of states, traces, etc
- $\subseteq$: comparison relation among concrete behaviors

Definition: abstract domain and abstraction relation

An **abstract domain** is defined by:

- $\mathbb{A}$: set of abstract behaviors
- $\sqsubseteq$: comparison relation that stands for logical strength

Moreover, an **abstraction relation** is a relation $(\models) \subseteq \mathbb{C} \times \mathbb{A}$ that describes **when a concrete behavior is correctly described by an abstract behavior**. It should satisfy:

- $\forall c \in \mathbb{C}, a_0, a_1 \in \mathbb{A},\ (c \models a_0 \wedge a_0 \sqsubseteq a_1) \Longrightarrow c \models a_1$
- $\forall c_0, c_1 \in \mathbb{C}, a \in \mathbb{A},\ (c_0 \subseteq c_1 \wedge c_1 \models a) \Longrightarrow c_0 \models a$

Example

Back to the **previous examples**:

- abstract predicate $\mathtt{x} \in [3, 7]$ describes all states that map $\mathtt{x}$ to a value comprised between 3 and 7
- abstract predicate $\mathtt{x} \geq 0$ describes all states that map x to a non-negative value
- abstract predicate $\mathtt{x} \in [3, 7]$ is stronger than abstract predicate $\mathtt{x} \geq 0$

Then:

- concrete behaviors: sets of integers, ordered by inclusion
- abstract elements: intervals, i.e., pairs $[a, b]$ with $a \leq b$;
- abstraction relation:

$$S \models [a, b] \quad \iff \quad \forall x \in S,\ a \leq x \leq b$$

Concretization

Are there more intuitive ways to specify $\models$? Yes, in most cases!

Definition: concretization function

A **concretization** is a function $\gamma : \mathbb{A} \longrightarrow \mathbb{C}$ that maps any abstract element to the largest concrete behavior that satisfies it (note: this is a strong property and γ may not exist for some $\models$!).

When it exists, the concretization is such that, for all $a \in \mathbb{A}$:

- $\gamma(a) \models a$
- $\forall c \in \mathbb{C},\ c \models a \Longrightarrow c \subseteq \gamma(a)$

Example:

$$\gamma : [a, b] \longmapsto \{x \mid a \leq x \leq b\}$$

Abstraction

Dual operation: go from concrete abstract elements to abstract ones.

Definition: abstraction function

An **abstraction** is a function $\alpha : \mathbb{C} \longrightarrow \mathbb{A}$ that maps any concrete behavior to the most precise abstract behavior that describes it (note: this is a strong property and α may not exist for some $\models$!).

When it exists, the abstraction is such that, for all $c \in \mathbb{C}$:

- $c \models \alpha(c)$
- $\forall a \in \mathbb{A},\ c \models a \Longrightarrow \alpha(c) \sqsubseteq a$

Example:

$$\alpha : S \longmapsto [\min S, \max S]$$

Galois connection

What if both abstraction and concretization functions exist?

They should **agree on the same abstraction relation**

it is then common to drop the abstraction relation and look only at α, γ:

Definition: Galois connection

A **Galois connection** is defined by a pair of orderings $(\mathbb{C}, \subseteq)$ and $(\mathbb{A}, \sqsubseteq)$, and a pair of functions $\alpha : \subseteq \longrightarrow \mathbb{A}$ and $\gamma : \sqsubseteq \longrightarrow \mathbb{C}$ such that:

$$\forall c \in \mathbb{C},\ \forall a \in \mathbb{A}, \qquad \alpha(c) \sqsubseteq a \qquad \Longleftrightarrow \qquad c \subseteq \gamma(a)$$

We write such a pair as follows: $(\mathbb{C}, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{A}, \sqsubseteq)$.

Example: $\mathbb{C} = \wp(E)$, $\mathbb{A} = \{\bot, \top\}$ (with the obvious orders), and:

$$\begin{array}{rclcrcl} \alpha(\emptyset) & = & \bot & \quad & \gamma(\bot) & = & \emptyset \\ \forall X \in E,\ X \neq \emptyset \Longrightarrow \alpha(X) & = & \top & & \gamma(\top) & = & E \end{array}$$

Galois connection properties

Galois connections describe **many abstraction relations** (but not all) and enjoy **many very useful algebraic properties**.

Assuming that α, γ form a Galois connection,

- α and γ are monotone
- $\forall c \in \mathbb{C},\ c \subseteq \gamma \circ \alpha(c)$
- $\forall a \in \mathbb{A},\ \alpha \circ \gamma(a) \sqsubseteq a$
- $\alpha \circ \gamma \circ \alpha = \alpha$
- $\gamma \circ \alpha \circ \gamma = \gamma$
- if both $\mathbb{C}$ and $\mathbb{A}$ have least upper bounds for any family of elements, then α preserves least upper bounds
- either function defines the other completely

Outline

Definition

A first family of abstractions/abstract domains:
abstractions of sets of values (i.e., the values a variable may take)

Value abstraction:

- $\mathbb{C} = \wp(\mathbb{V})$, e.g., the set of integers, or the set of machine integers, or the set of floating point values...
- $\subseteq$ is the set inclusion relation

Example: abstract sets of numeric values with their sign

- predicates: $\bot, \top, [\geq 0], [= 0], [\leq 0]$
- $\gamma(\bot) = \emptyset$, $\gamma(\top) = \mathbb{V}$, $\gamma([\geq 0]) = \{n \in \mathbb{V} \mid n \geq 0\}$, $\gamma([= 0]) = \{0\}$, $\gamma([\leq 0]) = \{n \in \mathbb{V} \mid n \leq 0\}$
- definition of α left as an exercise

Many other interesting examples...

Intervals

More interesting (and pratically useful) value abstrction: **intervals**
simply record a range that contains a set of scalar values

Abstract elements:

- $\perp$: empty set of values
- pairs $(n_0, n_1) \in \{-\infty\} \cup \mathbb{V} \times \mathbb{V} \times \{+\infty\}$ such that $n_0 \leq n_1$

Abstraction:

- $\alpha(\emptyset) = \perp$
- if $V \subseteq \mathbb{V}$ and $V \neq \emptyset$, then $\alpha(V) = (\min V, \max V)$

Concretization: maps a pair to the set of values in-between

Machine representation: pair of values

Useful to bound numerical computation, verify array bound checks...

A few numeric value abstractions

Constant values:

- abstract elements are $\perp, \top$, and elements of the form $[n]$ for any scalar value n
- the concretization of $[n]$ is $\{n\}$
- i.e., we keep precise information about exactly known values (singletons) and drop information about any other set of values

Congruence predicates:

- abstract elements are $\perp$ and pairs of the form (n, p) such that either $n = 0$ or $0 \leq p < n$
- the concretization of pair (n, p) is $\{kn + p \mid k \in \mathbb{Z}\}$
- such predicates are useful to discover information about pointer alignments or index arithmetic

Outline

Store abstraction

Value abstractions only describe sets of values,
but program semantics generally considers **sets of states**.

We recall:

- set of variables $\mathbb{X}$
- set of memory states $\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{V}$

Definition: store abstraction

A **state abstraction** defined by an abstract domain $(\mathbb{A}, \sqsubseteq)$ and an abstraction relation (i.e., abstraction function, concretization function or just abstraction relation) between $(\wp(\mathbb{M}), \subseteq)$ and $(\mathbb{A}, \sqsubseteq)$

We will study two ways of defining store abstractions:

1. from a value abstraction
2. directly

Definition of non relational abstractions

Data:

- value abstraction $(\mathbb{A}_{\mathcal{V}}, \sqsubseteq_{\mathcal{V}})$
- concretization function $\gamma_{\mathcal{V}} : \mathbb{A}_{\mathcal{V}} \to \wp(\mathbb{V})$

Definition: non relational abstraction

The **non-relational abstraction** is defined by

- the set of abstract elements $\mathbb{A}_{\mathcal{N}} = \mathbb{X} \to \mathbb{A}_{\mathcal{V}}$;
- the order relation $\sqsubseteq_{\mathcal{N}}$ defined by the pointwise extension of $\sqsubseteq_{\mathcal{V}}$
- the concretization function $\gamma_{\mathcal{N}}$

$$\begin{array}{rccl} \gamma_{\mathcal{N}} : & \mathbb{A}_{\mathcal{N}} & \longrightarrow & \wp(\mathbb{M}) \\ & M^{\sharp} & \longmapsto & \{ m \in \mathbb{M} \mid \forall \mathrm{x} \in \mathbb{X},\ m(\mathrm{x}) \in \gamma_{\mathcal{V}}(M^{\sharp}(\mathrm{x})) \} \end{array}$$

Exercise: determine the abstraction function when the underlying value abstraction also has one

Examples

A few concrete states:

$$\begin{array}{rcccc}
m_0: & \mathtt{x} \mapsto 25 & \mathtt{y} \mapsto 7 & \mathtt{z} \mapsto -12 \\
m_1: & \mathtt{x} \mapsto 28 & \mathtt{y} \mapsto -7 & \mathtt{z} \mapsto -11 \\
m_2: & \mathtt{x} \mapsto 20 & \mathtt{y} \mapsto 0 & \mathtt{z} \mapsto -10 \\
m_3: & \mathtt{x} \mapsto 35 & \mathtt{y} \mapsto 8 & \mathtt{z} \mapsto -9
\end{array}$$

An abstraction of $\{m_0, \ldots, m_3\}$, using the **interval abstract domain**:

$$\begin{array}{rccc}
M^\sharp: & \mathtt{x} & \mapsto & [25, 35] \\
 & \mathtt{y} & \mapsto & [-7, 8] \\
 & \mathtt{z} & \mapsto & [-12, -9]
\end{array}$$

Intuitions:

- each variable is treated separately
- the cosnstruction is parameterized by a value abstraction
- reduction: when one component is $\perp$, all should be

Outline

Limitations of non-relational abstraction

The name **"non-relational"** comes from the fact that it **cannot capture any relation** among program variables, such as

- if $x \geq 0$ then $y \geq 0$
- $0 \leq y \leq x$
- $2 * x - 3 * y + 8 = 0$

Whatever the value abstraction, applying non-relational abstraction will **result in a severe loss of precision**.

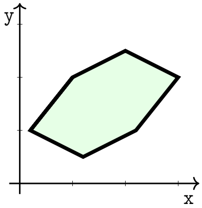
As opposed to **non-relational abstraction**, we can define **relational abstractions**, which can express some families of such constraints.

- such abstractions proceed directly from memory states (no intermediate step involving a value abstraction)
- they are generally more expressive, but often more complex/costly

Linear inequalities

Abstract domain of convex polyhedra:

- abstract states = conjunctions of linear inequalities
  e.g., $2x + y \leq 0 \wedge -x + 4y \leq 8$
- conretization: obvious function mapping abstract states into sets of points in the n dimension field that satisfy them (n: number of variables)



Two possible representations:

- symbolic, i.e., conjunction of linear inequalities
  (typically, some matrices)
- geometric, i.e., a set of vertices, edges, and rays

A specific characteristic: there is no best abstraction

Some other relational abstractions

Linear equalities:

- abstract elements stand for conjunctions of linear equalities among variables
- geometric interpretation: affine spaces in $\mathbb{V}^n$ (if n variables)
- representation of the form $A \cdot X = B$ where A and B are matrices
- algorithms come from linear algebra

Octagons:

- a restricted form of convex polyhedra, with constraints of the form $\pm x \pm y = c$ where x, y are variables and c is a constant
- in dimension 2, at most 8 faces, hence the name

Outline

Important points to remember, and what to learn next

Summary:

- orderings capture logical precision comparison
- the abstraction relation extends this to binding concrete/abstract elements
- abstraction: maps a concrete element to its best abstract approximation
- concretization: maps an abstract element to its concrete interpretation

What comes next ?

- algorithms to compute abstract elements that approximate the semantics, without running the programm

Construction of abstractions

There exist many ways of **constructing** sophisticated abstractions from basic ones:

- **(reduced) product**: express conjunctive properties
- **disjunctive completion**: express disjunctive properties
- **(reduced) cardinal power**: express conjunctions of implications

There are also many **specialized abstract domains**, e.g., for

- arrays
- inductive data-structures
- string buffers...

More in the book: chapters 5 and 8