Introduction to static analysis #3

Seminar @ Gondow Lab.

Goal of This Chapter

- The construction of a *static analysis framework*.
    - feature : general, can be used with different abstraction.
    - goal : compute program invariants by static abstraction
- How to construct a static analysis step by step.
    - We use basic programming language that operates over numerical states.

Outline of the book

- 3.1 : fix the language and its semantics.(6p)
- 3.2 : select an abstraction and fix their representation.(9p)
- 3.3 : derive the abstract semantics of programs from their semantics and abstractions.(18p)
- 3.4 : design of the interpreter.(2p)

Overview

- Semantics (3.1)
    - Programming Language
    - Concrete Semantics
        - Concrete Semantics
        - Properties of Interest
        - Input-Output Semantics
- Abstraction (3.2)
- Computable Abstract Semantics (3.3)
- Interpreter (3.4)

A Simple Programming Language (1/2)

We use simple programming language to illustrate the concepts of static analysis.

Some preparations:

- $\mathbb{X}$: a finite set of variable(which is fixed)
- $\mathbb{V}$: a set of scalar value
- $\mathbb{B}$: a set of boolean value
  - $\mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$

A Simple Programming Language (2/2)

Syntax of our language is:

- $n \quad \in \quad \mathbb{V}$
- $\mathrm{x} \quad \in \quad \mathbb{X}$
- $\odot \quad ::= \quad + \mid - \mid * \mid \ldots$
- $\lessdot \quad ::= \quad < \mid \leq \mid == \mid \ldots$
- $E \quad ::= \quad n \mid \mathrm{x} \mid E \odot E$
- $B \quad ::= \quad \mathrm{x} \lessdot n$
    - returns an element of $\mathbb{B}(= \{\mathbf{true}, \mathbf{false}\})$
- $C \quad ::= \quad \mathbf{skip} \mid C; C \mid \mathrm{x} := E \mid \mathbf{input}(\mathrm{x}) \mid \mathbf{if}(B)\{C\}\mathbf{else}\{C\}$
- $P \quad ::= \quad C$

A Simple Programming Language (3/2)

- $n \quad \in \quad \mathbb{V}$
  - scalar values
- $\mathbf{x} \quad \in \quad \mathbb{X}$
  - program variables
- $\odot \quad ::= \quad + \mid - \mid * \mid \ldots$
  - binary operators
- $\lessdot \quad ::= \quad < \mid \leq \mid == \mid \ldots$
  - comparison operators
- $E \quad ::= \quad n \mid \mathbf{x} \mid E \odot E$
  - scalar expressions

A Simple Programming Language (4/2)

- $B \quad ::= \quad \mathrm{x} \lessdot n$
  - returns an element of $\mathbb{B}(= \{\mathbf{true}, \mathbf{false}\})$
  - Boolean expressions
- $C \quad ::= \quad \mathbf{skip} \mid C; C \mid \mathrm{x} := E \mid \mathbf{input}(\mathrm{x}) \mid \mathbf{if}(B)\{C\}\mathbf{else}\{C\}$
  - commands
- $P \quad ::= \quad C$
  - program

Concrete Semantics

There're several kind of semantics. For instance, **trace semantics**, **denotational semantics**.

- **trace semantics** : describes program execution as a sequence of program state
- **denotational semantics** : describes only input-output relation

Before we can select which semantics to use, we discuss the family of properties of interest.

Properties of Interest

As in chapter 2, we focus on **reachability** properties.

Examples:

1. absence of run-time errors
2. verification of user assertions
   - execution should reach assertion point but should not meet the assertion condition

More general properties will be addressed in chapter 9.

Properties of Interest - reachability

Checking reachability properties would be:

1. pre-condition $\longrightarrow$ post-condition ($\longleftarrow$We need a semantic that capture this)
2. check post-condition

So we use ***input-output semantics***(one of denotational semantics).

An Input-Output Semantics

- Input-output semantics :
  - set of input states $\longmapsto$ set of output states
  - use mathematical function to map
  - output is a set of states because:
    - of the non-deterministic execution of **input**
    - we may observe infinitely many output states from one input
  - input is also a set of states
    - for the sake of compositionality

An Input-Output Semantics - compositionality

- Input-Output Semantics ***compositional***.

compositional : the semantics of a command can be defined by composing the semantics of its sub-commands.

e.g

> $C := C_1; C_2$

Semantics of C is defined by that of C_1 and C_2.

An Input-Output Semantics vs Interpreter

Input-output Semantics and Interpreter have much in common:

- input-output : set of input state**s** $\longmapsto$ set of output state**s**
- interpreter : a program and an input state $\longmapsto$ an output state

The main difference is:

- interpreter : inputs a ***single*** state and returns a ***single*** state

Essentially, interpreter implements the input-output semantics.

Memory States(1/2)

- *program state* should include:
    - *memory state* : contents of the memory
    - *control state* : a value of "program counter"(or next command to be executed)
- a state is defined by a memory state:
    - we use input-output semantics
    - input(output) state is fully determined by the contents of memory

Memory States(2/2)

- memory state $\mathbb{M}$ is defined by:
    - $\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{V}$

example:

- $\mathbb{X} = \{\mathrm{x}, \mathrm{y}\}$
    - $\mathbf{x} : 2, \mathbf{y} : 7$
- $m \in \mathbb{M}$ is:
    - $m = \{\mathrm{x} \mapsto 2, \mathrm{y} \mapsto 7\}$

📓 簡単に言うと、memory stateとは変数からスカラー値のマッピングのこと

Semantics of Scalar Expressions

How scalar expressions are evaluated.

- $[\![E]\!](m)$: semantics of expression E, in the memory state m.
  - $[\![E]\!] : \mathbb{M} \longrightarrow \mathbb{V}$
    - This is a function from memory states to scalar values

Semantics of each scalar expression is as follows:

- $[\![n]\!](m) = n$
- $[\![\mathrm{x}]\!](m) = m(\mathrm{x})$
  - $m(\mathrm{x})$: value of x in the memory state m
- $[\![E_0 \odot E_1]\!](m) = f_\odot([\![E_0]\!](m), [\![E_1]\!](m))$
  - $f_\odot$: mathematical function associated to the binary operator $\odot$

Semantics of Boolean Expressions

How Boolean expressions are evaluated.

- $[\![B]\!] = \mathbb{M} \longrightarrow \mathbb{B}$
  - This is a function from memory states to boolean values
- $[\![\mathrm{x} \lessdot n]\!] = f_{\lessdot}(m(\mathrm{x}), n)$
  - $f_{\lessdot}$: mathematical function associated to the comparison operator $\lessdot$

Semantics of Commands (1/6)

- $[\![C]\!]_{\mathscr{P}}$: semantics of a command C
  - a set of input states to a set of output states(which is observed **after** the command)
    - non-terminating executions are not observed
- $\wp(\mathbb{M})$: power set of memory states
  - intuitive explanation : "whether or not each variable is defined"
  - M : an element of $\wp(\mathbb{M})$, that is:
    - $M \in \wp(\mathbb{M})$

As a result, semantics of commands can be written as follows:

- $[\![C]\!]_{\mathscr{P}}$: $\wp(\mathbb{M}) \longrightarrow \wp(\mathbb{M})$

Semantics of Commands (2/6)

Semantics of commands is:

- $[\![\texttt{skip}]\!]_{\mathscr{P}}(M) = M$
  - identity function
- $[\![C_0; C_1]\!]_{\mathscr{P}}(M) = [\![C_1]\!]_{\mathscr{P}}([\![C_0]\!]_{\mathscr{P}}(M))$
  - composition of the semantics of each commands
- $[\![\mathrm{x} := E]\!]_{\mathscr{P}}(M) = \{m[\mathrm{x} \mapsto [\![E]\!](m)] \mid m \in M\}$
  - the evaluation of assignment updates the value of $\mathtt{x}$ in the memory states with the result of the evaluation of E.
- $[\![\texttt{input}(\mathrm{x})]\!]_{\mathscr{P}}(M) = \{m[\mathrm{x} \mapsto n] \mid m \in M, n \in \mathbb{V}\}$
  - replace the value of x with any possible scalar value n.

Quite easy.

Semantics of Commands (3/6)

Before we define semantics of **`if-else`** or **`while`**, we need some preparations.

- $\mathscr{F}_B$: filtering function. We need to define this first.
  - This function filter out memory states

Definition is as follows:

- $\mathscr{F}_B(M) = \{m \in M \mid [\![B]\!](m) = \mathbf{true}\}$
  - intuitive explanation : filter out memory states m in which B doesn't hold or can't be defined

Semantics of Commands (4/6)

Semantics of **`if-else`**:

- $[\![\texttt{if}(B)\{C_0\}\texttt{else}\{C_1\}]\!]_{\mathscr{P}}(M) = [\![C_0]\!]_{\mathscr{P}}(\mathscr{F}_B(M)) \cup [\![C_1]\!]_{\mathscr{P}}(\mathscr{F}_{\neg B}(M))$
  - union of the results of each branch

Semantics of Commands (5/6)

Semantics of **`while`**:

- $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\big(\cup_{i \geq 0}\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\big)$
  - complicated...

Let M_i be as follows:

- $M_i = \mathscr{F}_{\neg B}\big(([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B)^i(M)\big)$
  - intuitive explanation : B evaluates to **true** i times and to **false** for the last.
  - $[\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B$: filter memory states with B, then execute the command.

Semantics of Commands (6/6)

> Semantics of **`while`**:
>
> - $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\Big(\cup_{i\geq 0}\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\Big)$
>   - complicated...

Then, the set of output states would be $M_0 \cup M_1 \cup M_2 \ldots$, that is :

- $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \cup_{i\geq 0} M_i = \cup_{i\geq 0}\mathscr{F}_{\neg B}\Big(\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\Big)$

$\mathscr{F}_B$ commutes with the union, thus:

- $\cup_{i\geq 0} M_i = \mathscr{F}_{\neg B}\Big(\cup_{i\geq 0}\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\Big)$

Therefore,

- $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\Big(\cup_{i\geq 0}\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\Big)$

Overview

- Semantics (3.1)
- Abstraction (3.2)
  - The concept of abstraction
  - Non-relational abstraction
  - Relational abstraction
- Computable Abstract Semantics (3.3)
- Interpreter (3.4)

Concrete, Abstract

We carefully distinguish between these:

- domain the program is defined ($\longrightarrow$ "***concrete***" qualifier for this)
- domain that is used for the analysis of program ($\longrightarrow$ "***abstract***" qualifier for this)

Concrete Domain

Definition: Concrete Domain

- a set $\mathbb{C}$: concrete domain, describes concrete behaviors
- $\subseteq$: order relation, compares program behaviors in the logical point of view
  - $x \subseteq y$ means that x implies behavior y, that is:
    - x expresses a stronger property than y.

Example:

- $\mathbb{C} = \wp(\mathbb{M})$
  - $c \in \mathbb{C}, c = \{\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2\}$

Abstract Domain (1/3)

Some preparations:

- c : concrete element
- a : abstract element
- $c \models a$: c satisfies the logical properties expressed by a

Abstract Domain (2/3)

Definition: Abstract Domain and Abstract Relation

- **abstract domain** : a pair of a set $\mathbb{A}$ and an ordering relation $\sqsubseteq$ over that set.

Given a concrete domain $(\mathbb{C}, \subseteq)$, **abstraction** is defined by:

- $(\mathbb{A}, \sqsubseteq)$
- an abstract relation "$\vDash$" such that:
  - for all $c \in \mathbb{C}, a_0, a_1 \in \mathbb{A}$, if $c \vDash a_0$ and $a_0 \sqsubseteq a_1$, then $c \vDash a_1$; and
  - for all $c_0, c_1 \in \mathbb{C}, a \in \mathbb{A}$, if $c_0 \subseteq c_1$ and $c_1 \vDash a$, then $c_0 \vDash a$.

Abstract Domain (3/3)

Example 3.2 (Abstraction) :

- concrete domain : $\wp(\mathbb{M})$
- variable : x, y

Elements of concrete domain :

- $M_0 = \{m \in \mathbb{M} \mid 0 \leq m(\mathrm{x}) < m(\mathrm{y}) \leq 8\}$
- $M_1 = \{m \in \mathbb{M} \mid 0 \leq m(\mathrm{x})\}$

An element of abstract domain :

- $M^\sharp$: over-approximates each value
  - $\mathbf{x}$: [0, 10]
  - $\mathbf{y}$: [0, 100]

Concretization Function (1/n)

Sometimes, " $\models$ " is not useful. Thus, we define concretization function.

Definition 3.3 (Concretization function)

A concretization function (or, for short, concretization) :

- $\gamma : \mathbb{A} \to \mathbb{C}$
  - for any abstract element a, $\gamma(a)$ satisfies a. ($\gamma(a) \models a$)
  - $\gamma(a)$ is the maximum element of $\mathbb{C}$ that satisfies a

Concretization Function (2/n)

- A concretization function fully describe the abstraction relation:
  - $\forall c \in \mathbb{C}, \forall a \in \mathbb{A}, \qquad c \vDash a \iff c \subseteq \gamma(a)$
- Concretization function is also monotone.

Example 3.3 (Concretization function)

- same notion as example 3.2. $(M^\sharp, M_0, M_1)$
- There are memory states in $\gamma(M^\sharp)$ that are not in M_1

> - $M_1 \nvDash M^\sharp$: $(11, 0)$ is an element of M_1, but doesn't satisfy $M^\sharp$

Abstraction Function (1/3)

Definition 3.4 (Abstraction function)

c has a **best abstraction** if and only if there exists a such that:

- a is an abstraction of c
- any other abstraction of c is greater than c.

Abstraction function (or for short, abstraction):

- $\alpha : \mathbb{C} \rightarrow \mathbb{A}$
  - This function maps each concrete element to its best abstraction

Abstraction function is:

- the dual of concretization function
- monotone

Abstraction Function (2/3)

Example 3.4 (Abstraction function)

- same notion as example 3.2 and 3.3
- $M^\sharp$ is not a best abstraction of M_0
  - Best abstraction of M_0 is smaller than $M^\sharp$

> - $M_0 = \{m \in \mathbb{M} \mid 0 \leq m(\mathrm{x}) < m(\mathrm{y}) \leq 8\}$
> - $M_1 = \{m \in \mathbb{M} \mid 0 \leq m(\mathrm{x})\}$
> - $M^\sharp$: over-approximates each value
>   - x : [0, 10]
>   - y : [0, 100]

Abstraction Function (3/3)

Note:

- The existence of a best abstraction is not guaranteed in general.
- Abstract relations such that no concretization function can be defined will not arise in this book.

Galois Connection (1/3)

When an abstraction relation defines both

- concretization function
- abstraction function

they are tightly related to each other (which we call ***Galois connection***).

Galois Connection (2/3)

Definition 3.5 (Galois connection):

Galois connection is a pair made of a concretization function γ and an abstraction function α such that:

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

We write such a pair as follows:

- $(\mathbb{C}, \subseteq) \underset{\gamma}{\overset{\alpha}{\leftrightarrows}} (\mathbb{A}, \sqsubseteq)$

Galois Connection (3/3)

Some interesting properties (proof is in B.1):

- α and γ are monotone function.
- $\forall c \in \mathbb{C}$
  - $c \subseteq \gamma(\alpha(c))$
  - applying the abstraction function and concretizing the result back yield a less precise result
- $\forall a \in \mathbb{A}$
  - $\alpha(\gamma(a)) \sqsubseteq a$
  - concretizing an abstract element and abstracting the result back refines the information available in the initial abstract element (which is known as *reduction*)

Overview

- Semantics (3.1)
- Abstraction (3.2)
  - The concept of abstraction
  - Non-relational abstraction
  - Relational abstraction
- Computable Abstract Semantics (3.3)
- Interpreter (3.4)

(Non-relational / Relational) Abstraction

- Non-relational : それぞれの変数を独立に抽象化する
- Relational : 変数間の関係も含めて抽象化する (relationalが表すとおり)

Non-relational Abstraction

Non-relational abstraction proceeds in two steps:

1. For each variable, it collects the values that the variable may take.
2. Then, over-approximates each of these set of values with one abstract element per variable (*value abstraction*).

Value Abstraction (1/n)

Definition 3.6 (Value abstraction)

A **value abstraction** is an abstraction of $(\wp(\mathbb{V}), \subseteq)$

As we saw in chapter 2, *interval* and *sign* constraints define value abstractions.

Value Abstraction (2/n)

Example 3.5 (Signs) (Figure 3.5)

- sign abstraction domain $\mathbb{A}_{\mathscr{S}}$: $[\geq 0], [\leq 0], [= 0]$
  - $\top$: any set of values
  - $\bot$: empty set of values
- concretization function
  - $\gamma_{\mathscr{S}}$:
    - $[\geq 0] \quad \longmapsto \quad \{n \in \mathbb{V} \mid n \geq 0\}$
    - $[\leq 0] \quad \longmapsto \quad \{n \in \mathbb{V} \mid n \leq 0\}$
    - $[= 0] \quad \longmapsto \quad \{0\}$
    - $\top \quad \longmapsto \quad \mathbb{V}$
    - $\bot \quad \longmapsto \quad \emptyset$

Value Abstraction (3/n)

Example 3.6 (A variation on the lattice of sign, with no abstraction function)

- If we remove $[=0]$ from the abstract domain above, it doesn't have best abstract function.
- concrete set $\{0\}$
  - we can't define abstraction function of this
  - $[\leq]$ and $[\geq]$ are incomparable

As a consequence:

- in general, it is impossible to identify one element as a most precise (sound) one.

> Provided the analysis designer and user are aware of this fact, it is not a serious limitation, however.

Value Abstraction (4/n)

Example 3.7 (Intervals) (Figure 3.5)

- intervals value abstract domain $\mathbb{A}_{\mathscr{I}}$:
    - $\perp$: the empty set of values
    - (n_0, n_1) :
        - n_0 : either $-\infty$ or a value
        - n_1 : either $+\infty$ or a value
        - $n_0 \leq n_1$
- concretization function :
    - $\gamma_{\mathscr{I}}$:
        - $\perp \longmapsto \emptyset$
        - $[n_0, n_1] \quad \longmapsto \quad \{n \in \mathbb{V} \mid n_0 \leq n \leq n_1\}$
        - $[n_0, +\infty] \quad \longmapsto \quad \{n \in \mathbb{V} \mid n_0 \leq n\}$

Value Abstraction (5/n)

Example 3.8 (Congruences)

- abstract domain of congruences :
  - describes sets of values using congruence relations
- abstract element :
  - $\perp$: empty set of values
  - (n, p) : set of values that are equal to n modulo p.
    - $p = 0$ or $0 \leq n < p$
- concretization function :
  - $\gamma_{\mathscr{C}}$:
    - $\perp \quad \longmapsto \quad \emptyset$
    - $(n, p) \quad \longmapsto \quad \{n + kp \mid k \in \mathbb{Z}\}$

Also,

Non-relational Abstraction (1/4)

Definition 3.7 (Non-relational abstraction)

Assume that a value abstraction is given, that is

- a value abstraction : $(\mathbb{A}_{\mathscr{V}}, \sqsubseteq)$
- concretization function $\gamma_{\mathscr{V}} : \mathbb{A}_{\mathscr{V}} \to \wp(\mathbb{V})$
- a least element : $\perp_{\mathscr{V}}$
- a greatest element : $\top_{\mathscr{V}}$

Then, non-relational abstraction is is defined by

- set of abstract elements $\mathbb{A}_{\mathscr{N}} = \mathbb{X} \to \mathbb{A}_{\mathscr{V}}$
- order relation $\sqsubseteq_{\mathscr{A}}$: defined by
- point-wise extension of $\sqsubseteq_{\mathscr{V}}$
- $M_0^\sharp \sqsubseteq_{\mathscr{N}} M_1^\sharp$ if and only if $\forall \mathbf{x} \in \mathbb{X},\ M_0^\sharp(\mathbf{x}) \sqsubseteq_{\mathscr{V}} M_1^\sharp(\mathbf{x})$

Non-relational Abstraction (2/4)

Intuitive explanation:

- treats each variable independently
  - applies the value abstraction to each variable separately from the other
- order relation is point-wise

The ***least element*** of the non-relational abstract domain is

- the function that maps each variable to the least element $\perp_{\mathscr{V}}$:
  - $\forall \mathbf{x} \in \mathbb{X}, \perp_{\mathscr{N}}(\mathbf{x}) = \perp_{\mathscr{V}}$

The ***greatest element*** $\top_{\mathscr{N}}$ can be defined similarly.

Non-relational Abstraction (3/4)

- When the value abstraction has an abstraction function $\alpha_{\mathscr{V}}$:
  - the non-relational abstraction also has one.

It is defined as follows:

- $\alpha_{\mathscr{N}} : M \longmapsto \Big((\mathbf{x} \in \mathbb{X}) \longmapsto \alpha_{\mathscr{V}}(\{m(\mathbf{x}) \mid m \in M\}) \Big)$

Note:

- $\bot_{\mathscr{N}}$ is the best abstraction of $\emptyset$

Non-relational Abstraction (4/4)

Example 3.9 (Non-relational abstraction)

Assumption:

- $\mathbb{X} = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$
- memory states
  - $m_0 : \quad \mathrm{x} \mapsto 25 \quad \mathrm{y} \mapsto 7 \quad \mathrm{z} \mapsto -12$
  - $m_1 : \quad \mathrm{x} \mapsto 28 \quad \mathrm{y} \mapsto -7 \quad \mathrm{z} \mapsto -11$
  - $m_2 : \quad \mathrm{x} \mapsto 20 \quad \mathrm{y} \mapsto 0 \quad \mathrm{z} \mapsto -10$
  - $m_3 : \quad \mathrm{x} \mapsto 35 \quad \mathrm{y} \mapsto 8 \quad \mathrm{z} \mapsto -9$

The best abstraction of $\{m_0, m_1, m_2, m_3\}$ can be defined as follows :

- With the signs abstraction :
  - $M^\sharp : \quad \mathrm{x} \mapsto \quad\quad \mathrm{y} \mapsto \quad\quad \mathrm{z} \mapsto$

Overview

- Semantics (3.1)
- Abstraction (3.2)
  - The concept of abstraction
  - Non-relational abstraction
  - Relational abstraction
    - linear equalities
    - convex polyhedra
    - octagons
- Computable Abstract Semantics (3.3)
- Interpreter (3.4)

Relational Abstraction (1/4)

Such as *convex polyhedra*.

Definition 3.8 (Linear equalities)

- The elements of abstract domain of linear equalities :
  - $\perp$: empty set
  - conjunctions of linear equality constraints : constrain sets of memory states.
    - such as $\mathrm{y} = a\mathrm{x}$

In the geometrical point of view :

- abstract elements are in the affine space $\mathbb{V}^N$
  - N : dimension (number of variables)

This abstraction features :

Relational Abstraction (2/4)

Definition 3.8 (Convex polyhedra)

- elements of abstract domain of linear inequalities :
  - $\bot$: empty set
  - conjunctions of linear **in**equality constraints : constrain sets of memory states.

In the geometrical point of view :

- abstract elements : convex polyhedra of all dimension in $\mathbb{V}^N$
  - N : dimension (number of variables)

This abstraction features :

- concretization
- but no best abstraction function
  - certain concrete sets do have a best abstraction though

Relational Abstraction (3/4)

Definition 3.9 (Octagons)

- element of abstract domain of octagons :
  - $\perp$: empty set
  - conjunctions of linear inequality constraints of the form below:
    - $\pm \mathbf{x} \pm \mathbf{y} \leq c$
    - $\pm x = c$

In the geometrical point of view :

- abstract elements : "octagonal" shape

This abstraction features:

- best abstraction function

Relational Abstraction (4/4)

- It is difficult to decide which abstract domain describes relational constraints efficiently.
  - We will not discuss this topic any further.

Overview

- Semantics (3.1)
- Abstraction (3.2)
- Computable Abstract Semantics (3.3)
  - introduction
  - semantics of each commands
  - soundness
- Interpreter (3.4)

Computable Abstract Semantics (1/3)

- we **use non-relational** abstract domain
  - we also discuss the modifications which is required to use relational abstract domain.

The form of analysis is :

- mathematical function
  - input : a program and an abstract pre-condition
  - output : an abstract post-condition

Computable Abstract Semantics (2/3)

Some preparations :

- $\mathbb{A}$: the state abstract domain
- γ : associated concretization function
  - $\mathbb{A}_{\mathscr{V}}$: underlying value abstraction.
  - $\gamma_{\mathscr{V}}$: concretization function

The design of the analysis aims at:

- the soundness in the sense of definition 2.6
  - See figure 3.7
    - $[\![\mathrm{p}]\!]^{\sharp}_{\mathscr{P}}$: the static analysis function (or *abstract semantics*)

Computable Abstract Semantics (3/3)

$$\begin{tikzcd} a_{\text{pre}} \arrow[r, "\text{analyze}\ \texttt{p}"] & [\![\texttt{p}]\!]^{\sharp}_{\mathscr{P}}(a_{\text{pre}}) \\ m \arrow[u, "\in\gamma(\cdot)"] \arrow[r, "\text{run}\ \texttt{p}"] & m' \end{tikzcd}$$

$$\begin{tikzcd} a_{\text{pre}} & a_{\text{post}} \\ m \arrow[u, "\in\gamma(\cdot)"] \arrow[r, "\text{run}\ \texttt{p}"] & m' \arrow[u, "\in\gamma(\cdot)"'] \end{tikzcd}$$

- $[\![\texttt{p}]\!]^{\sharp}_{\mathscr{P}}$: analysis function, or *abstract semantics*

Overview

- Semantics (3.1)
- Abstraction (3.2)
- Computable Abstract Semantics (3.3)
  - introduction
  - semantics of each commands
  - soundness
- Interpreter (3.4)

Abstract Semantics of Each Commands

We're going to define the semantics of $[\![\cdot]\!]^{\sharp}_{\mathscr{P}}$ by induction.

- Definition of the semantics : very similar to that of concrete semantics.
- Soundness : ensured
  - soundness is ensured in a inductive manner
- Abstract semantics of a command : defined by that of its sub-commands.

That's all 🙂

- $[\![n]\!]^{\sharp}(M^{\sharp}) = \phi_{\mathscr{V}}(n)$
- $[\![\mathrm{x}]\!]^{\sharp}(M^{\sharp}) = M^{\sharp}(\mathrm{x})$
- $[\![\mathtt{E}_0 \odot \mathtt{E}_1]\!]^{\sharp}(M^{\sharp}) = f^{\sharp}_{\odot}([\![\mathtt{E}_0]\!]^{\sharp}(M^{\sharp}), [\![\mathtt{E}_1]\!]^{\sharp}(M^{\sharp}))$
- $[\![\mathtt{C}]\!]^{\sharp}_{\mathscr{P}}(\bot) = \bot$
- $[\![\mathtt{skip}]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = M^{\sharp}$
- $[\![\mathtt{C}_0; \mathtt{C}_1]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = [\![\mathtt{C}_0]\!]^{\sharp}_{\mathscr{P}}([\![\mathtt{C}_1]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}))$
- $[\![\mathrm{x} := \mathtt{E}]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = M^{\sharp}[\mathrm{x} \mapsto [\![\mathtt{E}]\!]^{\sharp}(M^{\sharp})]$
- $[\![\mathtt{input}(\mathrm{x})]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = M^{\sharp}[\mathrm{x} \mapsto \top_{\mathscr{V}}]$
- $[\![\mathtt{if}(B)\{C_0\}\mathtt{else}\{C_1\}]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = [\![C_0]\!]^{\sharp}_{\mathscr{P}}(\mathscr{F}^{\sharp}_{B}(M^{\sharp})) \sqcup^{\sharp} [\![C_1]\!]^{\sharp}_{\mathscr{P}}(\mathscr{F}^{\sharp}_{\neg B}(M^{\sharp}))$
- $[\![\mathtt{while}(B)\{C\}]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = \mathscr{F}^{\sharp}_{\neg B}(\mathrm{abs\_iter}([\![C]\!]^{\sharp}_{\mathscr{P}} \circ \mathscr{F}^{\sharp}_{B}, M^{\sharp}))$

Bottom Element, Skip Commands

Bottom Element

- $[\![\texttt{C}]\!]^{\sharp}_{\mathscr{P}}(\bot) = \bot$
  - intuitive explanation : running a program from empty set of states is empty.
  - soundness : ensured

Skip Commands

- $[\![\mathbf{skip}]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = M^{\sharp}$
  - input is not modified
  - soundness : ensured

Sequences of Commands

- Soundness property of figure 3.7 is stable is under composition.
  - $[\![\mathbf{p}_0 ; \mathbf{p}_1]\!]_{\mathscr{P}}(M) = [\![\mathbf{p}_0]\!]_{\mathscr{P}}([\![\mathbf{p}_1]\!]_{\mathscr{P}}(M))$

Sequences of Commands

- $[\![\mathtt{C}_0 ; \mathtt{C}_1]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}) = [\![\mathtt{C}_0]\!]^{\sharp}_{\mathscr{P}}([\![\mathtt{C}_1]\!]^{\sharp}_{\mathscr{P}}(M^{\sharp}))$
- this equation ensures that we can prove soundness by induction.

Approximation of Composition (1/n)

Theorem 3.1 (Approximation of composition)

- $F_0, F_1 : \wp(\mathbb{M}) \to \wp(\mathbb{M})$
  - two monotone functions
- $F_0^\sharp, F_1^\sharp : \mathbb{A} \to \mathbb{A}$
  - these two functions over-approximate the two function above.
  - such that
    - $F_0 \circ \gamma \subseteq \gamma \circ F_0^\sharp$ and $F_1 \circ \gamma \subseteq \gamma \circ F_1^\sharp$
- then, $F_0 \circ F_1$ can be over-approximated by $F_0^\sharp \circ F_1^\sharp$

Approximation of Composition (2/n)

Proof

- Assumption : $M^\sharp \in \mathbb{A}$
- $F_1 \circ \gamma(M^\sharp) \subseteq \gamma \circ F_1^\sharp(M^\sharp)$ (by the soundness of F_1)
- $F_0 \circ F_1 \circ \gamma(M^\sharp) \subseteq F_0 \circ \gamma \circ F_1^\sharp(M^\sharp)$ (applied F_0, since F_0 is monotone)
  - $\subseteq \gamma \circ F_0^\sharp \circ F_1^\sharp(M^\sharp)$ (by the soundness of F_0)
- then,
  - $F_0 \circ F_1 \circ \gamma(M^\sharp) \subseteq \gamma \circ F_0^\sharp \circ F_1^\sharp(M^\sharp)$
- so, $F_0 \circ F_1$ is over-approximated by $\circ F_0^\sharp \circ F_1^\sharp$

Note:

- concrete semantics heavily relies on this composition of function.

Expressions (1/5)

Abstract Interpretation of Expressions

- $[\![\mathbf{E}]\!]^\sharp$: abstract interpretation of expressions
- $[\![\mathbf{E}]\!]^\sharp : \mathbb{A} \to \mathbb{A}_{\mathscr{V}}$
- semantics of expressions
- $[\![n]\!]^\sharp(M^\sharp) = \phi_{\mathscr{V}}(n)$
- $[\![\mathrm{x}]\!]^\sharp(M^\sharp) = M^\sharp(\mathrm{x})$
- $[\![\mathbf{E}_0 \odot \mathbf{E}_1]\!]^\sharp(M^\sharp) = f^\sharp_\odot([\![\mathbf{E}_0]\!]^\sharp(M^\sharp), [\![\mathbf{E}_1]\!]^\sharp(M^\sharp))$
- soundness : ensured
- we will not see the proof though.

Expressions (2/5)

- $[\![n]\!]^\sharp(M^\sharp) = \phi_{\mathscr{V}}(n)$
  - This shoud return any abstract element that over-approximate n
  - If the value abstraction has a best abstraction $\alpha_{\mathscr{V}}$, $\alpha_{\mathscr{V}}(\{n\})$ is enough.
  - $\phi_{\mathscr{V}} : \mathbb{V} \to \mathbb{A}_{\mathscr{V}}$
    - This function may not return the most precise abstraction.
    - This function is such that $n \in \gamma_{\mathscr{V}}(\phi_{\mathscr{V}}(n))$

Expressions (3/5)

- $[\![\mathrm{x}]\!]^\sharp(M^\sharp) = M^\sharp(\mathrm{x})$
  - simply return a abstraction that is associated to the variable.

> - set of abstract elements $\mathbb{A}_{\mathcal{N}} = \mathbb{X} \to \mathbb{A}_{\mathcal{V}}$

Expressions (4/5)

- $[\![\mathtt{E}_0 \odot \mathtt{E}_1]\!]^\sharp(M^\sharp) = f^\sharp_\odot([\![\mathtt{E}_0]\!]^\sharp(M^\sharp), [\![\mathtt{E}_1]\!]^\sharp(M^\sharp))$
  - we need to apply the conservative abstraction of $f_\odot$ in the non-relational lattice.
  - we need an operator $f^\sharp_\odot$ such that:
    - for all $n_0^\sharp, n_1^\sharp \in \mathbb{A}_\mathscr{V}$
      - $\{f_\odot(n_0, n_1) \mid n_0 \in \gamma_\mathscr{V}(n_0^\sharp)$ and $n_1 \in \gamma_\mathscr{V}(n_1^\sharp)\} \subseteq \gamma_\mathscr{V}(f^\sharp_\odot(n_0^\sharp, n_1^\sharp))$
  - $f^\sharp_\odot$ shoud over-approximate the effect of operation of $f_\odot$ on concrete value.

Expressions (5/5)

Example 3.10 (Abstract semantics of expressions)

- we use interval abstraction
- $M^\sharp$ is defined by $M^\sharp(\mathrm{x}) = [10, 20]$ and $M^\sharp(\mathrm{y}) = [8, 9]$

Interpretation of $\mathrm{x} + 2 * \mathrm{y} - 6$: ($f^\sharp_-$ and $f^\sharp_+$ can be used)

- $[\![\mathrm{x} + 2 * \mathrm{y} - 6]\!]^\sharp(M^\sharp)$
    - $= f^\sharp_-([\![\mathrm{x} + 2 * \mathrm{y}]\!]^\sharp(M^\sharp), [\![6]\!]^\sharp(M^\sharp))$

Assignments (1/n)

> $$[\![\mathrm{x} := E]\!]_{\mathscr{P}}(M) = \{m[\mathrm{x} \mapsto [\![E]\!](m)] \mid m \in M\}$$

Recall that assignment is the composition of

1. Evaluation of the expression $\mathtt{E}$ to n
2. Update of the variable $\mathtt{x}$ with n

This composition can be over-approximated piece by piece (Theorem 3.1).

Assignments (, Input) (2/n)

Assignments

- target : $\texttt{x} := \texttt{E}$
- $[\![\mathrm{x} := \texttt{E}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto [\![\texttt{E}]\!]^\sharp(M^\sharp)]$

input

- $[\![\texttt{input}(\mathrm{x})]\!]^\sharp_{\mathscr{P}}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto \top_{\mathscr{V}}]$
  - repaleced the value with $\top_{\mathscr{V}}$

Assignments (3/n)

Example 3.11 (Analysis of an assignment command)

> - $M^\sharp(\mathrm{x}) = [10, 20]$ and $M^\sharp(\mathrm{y}) = [8, 9]$
> - $[\![\mathrm{x} + 2 * \mathrm{y} - 6]\!]^\sharp(M^\sharp) = [20, 32]$

- $[\![\mathrm{x} := \mathrm{x} + 2 * \mathrm{y} - 6]\!]^\sharp(M^\sharp) =$

Assignments (with Relational Abstract Domain) (1/n)

Analysis of Assignments Using a Relational Abstract Domain

> 1. Add temporary dimension x' that is meant to describe the value of the expression
> 2. Represent as precisely as possible the constraint $\mathrm{x}' = \mathbf{E}$
> 3. Project out dimension x, and rename x' to x

Assignments (with Relational Abstract Domain) (2/n)

Example 3.12

Assumption:

- abstract domain : convex polyhedra
- abstract pre-condition : $2 \leq \mathrm{x} \leq 3 \land 1 - \mathrm{x} \leq \mathrm{y}$
- assignment : $\mathrm{x} := \mathrm{y} + \mathrm{x} + 2$

We introduce the variable x' and write the constraint as below:

- $2 \leq \mathrm{x} \leq 3 \land 1 - \mathrm{x} \leq \mathrm{y} \land \mathrm{x}' = \mathrm{y} + \mathrm{x} + 2$

From the last term, we get $\mathrm{x} = \mathrm{x}' - \mathrm{y} - 2$. Then, apply this formula and we get

- $2 \leq \mathrm{x}' - \mathrm{y} - 2 \leq 3 \land 3 - \mathrm{x}' + \mathrm{y} \leq \mathrm{y}$
- $\iff 4 \leq \mathrm{x}' - \mathrm{y} \leq 5 \land 3 \leq \mathrm{x}'$ (rename x' to x if you want)

Conditional Branching

- An in the last paragraph, we over-approximate the definition of concrete semantics step-by-step.

> $$[\![\texttt{if}(B)\{C_0\}\texttt{else}\{C_1\}]\!]_{\mathscr{P}}(M) = [\![C_0]\!]_{\mathscr{P}}(\mathscr{F}_B(M)) \cup [\![C_1]\!]_{\mathscr{P}}(\mathscr{F}_{\neg B}(M))$$

We will follow these steps:

1. design an operation to over-approximate $\mathscr{F}_B$ for any Boolean expression B.
2. use the abstract semantics of both branches
3. apply the over-approximation of the union of concrete sets.

Analysis of Condition (1/4)

Analysis of Conditions

- abstraction of filtering function $\mathscr{F}_B$, which we denote by $F_B^\sharp$
- $\mathscr{F}_B$
  - input : memory states
  - output : memory states such that B evaluates to *true*.
- $\mathscr{F}_B^\sharp$
  - input : an abstract state
  - output : an abstract state refined by the condition B

$\mathscr{F}_B^\sharp$ should satisfies the following soundness condition (ref. figure 3.7):

- for all conditions B and all abstract states $M^\sharp$
  - $\mathscr{F}_B(\gamma(M^\sharp)) \subseteq \gamma(\mathscr{F}_B^\sharp(M^\sharp))$

Analysis of Condition (2/4)

We will see some examples.

- Sign abstract domain $\{\bot, \top, [= 0], [\geq 0], [\leq 0]\}$
  - $\mathscr{F}^\sharp_{\mathrm{x}<0}(M^\sharp) =$
    - $(\mathbf{y} \in \mathbb{X}) \mapsto \bot$ if $M^\sharp(\mathbf{x}) = [\geq 0]$ or $[= 0]$ or $\bot$
    - $M^\sharp[\mathbf{x} \mapsto [\leq]]$ if $M^\sharp(\mathbf{x}) = [\leq 0]$ or $\top$
- Interval abstract domain $M^\sharp(\mathbf{x}) = [a, b]$
  - $\mathscr{F}^\sharp_{\mathrm{x}\leq n}(M^\sharp) =$
    - $(\mathbf{y} \in \mathbb{X}) \mapsto \bot$ if $a > n$
    - $M^\sharp[\mathbf{x} \mapsto [a, n]]$ if $a \leq n \leq b$
    - $M^\sharp$ if $b \leq n$

Analysis of Condition (3/4)

Example 3.13 (Analysis of a condition)

We consider the code fragment below that computes the absolute value of $\mathrm{x} - 7$.

```
if(x > 7){
    y := x - 7
}else{
    y := 7 - x
}
```

Assumption:

- pre-condition $M^\sharp : \mathrm{x} \mapsto \top, \mathrm{y} \mapsto \top$

Then, by the rule above,

- $\mathscr{F}_{\mathrm{x}>7}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto [8, +\infty)]$
- $\mathscr{F}_{\mathrm{x}\leq 7}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto (-\infty, 7]]$

Analysis of Condition (4/4)

Theorem 3.3 (Soundness of the abstract interpretation conditions)

- for all...
  - expressions B
  - non-relational abstract elements $M^\sharp$
  - memory states m such that $m \in \gamma(M^\sharp)$
- if $[\![B]\!](m) = \mathbf{true}$, then $m \in \gamma(\mathscr{F}^\sharp_B(M^\sharp))$

Analysis of Flow Joins (1/3)

> $[\![\texttt{if}(B)\{C_0\}\texttt{else}\{C_1\}]\!]_{\mathscr{P}}(M) = [\![C_0]\!]_{\mathscr{P}}(\mathscr{F}_B(M)) \cup [\![C_1]\!]_{\mathscr{P}}(\mathscr{F}_{\neg B}(M))$

Next, we want to abstract the union operator $\cup$.
Let $\sqcup^\sharp$ be the abstract union (join) operator.

$\sqcup^\sharp$ should satisfy the following soundness property:

Theorem 3.4 (Soundness of abstract join)

Let $M_0^\sharp$ and $M_1^\sharp$ be the two abstract states.

- $\gamma(M_0^\sharp) \cup \gamma(M_1^\sharp) \subseteq \gamma(M_0^\sharp \sqcup^\sharp M_1^\sharp)$

Analysis of Flow Joins (2/3)

To define $\sqcup^\sharp$, we can simply

- define a join operator $\sqcup^\sharp_{\mathscr{V}}$ in the value abstract domain.
- apply operator $\sqcup^\sharp_{\mathscr{V}}$ in a point-wise manner:
  - for all variable $\mathbf{x}$, $(M_0^\sharp \sqcup^\sharp M_1^\sharp)(\mathbf{x}) = M_0^\sharp(\mathbf{x}) \sqcup^\sharp_{\mathscr{V}} M_1^\sharp(\mathbf{x})$

The definition of $\sqcup^\sharp_{\mathscr{V}}$ depends on the abstract domain.
For instance, for the interval domain:

- $[a_0, b_0] \sqcup^\sharp_{\mathscr{V}} [a_1, b_1] = [\mathtt{min}(a_0, b_0), \mathtt{max}(a_1, b_1)]$
- $[a_0, b_0] \sqcup^\sharp_{\mathscr{V}} [a_1, +\infty) = [\mathtt{min}(a_0, b_0), +\infty)$

Analysis of Flow Joins (3/3)

Example 3.14 (Analysis of flow joins)

- $M_0^\sharp = \{\mathrm{x} \mapsto [0,3], \mathrm{y} \mapsto [6,7], \mathrm{z} \mapsto [4,8]\}$
- $M_1^\sharp = \{\mathrm{x} \mapsto [5,6], \mathrm{y} \mapsto [0,2], \mathrm{z} \mapsto [6,9]\}$

Then,

- $M_0^\sharp \cup^\sharp M_1^\sharp = \{\mathrm{x} \mapsto [\quad,\quad], \mathrm{y} \mapsto [\quad,\quad], \mathrm{z} \mapsto [\quad,\quad]\}$

Analysis of Conditional Commands (1/3)

Now, we have defined

- condition
- flow joins

and we can use those to define the semantics of conditional commands.

Semantics of conditional commands:

- $[\![\mathtt{if}(B)\{C_0\}\mathtt{else}\{C_1\}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = [\![C_0]\!]^\sharp_{\mathscr{P}}(\mathscr{F}^\sharp_B(M^\sharp)) \sqcup^\sharp [\![C_1]\!]^\sharp_{\mathscr{P}}(\mathscr{F}^\sharp_{\neg B}(M^\sharp))$

This definition is very similar to that of concrete one.

Analysis of Conditional Commands (2/3)

We use this program from example 3.13 here.

```
if(x > 3){
    y := x - 3
}else{
    y := 3 - x
}
```

Analysis of Conditional Commands (3/3)

Example 3.15 (Analysis of a conditional command)

- abstract pre-condition : $M^\sharp = \{\mathrm{x} \mapsto \top, \mathrm{y} \mapsto \top\}$

Analysis proceeds as follows :

1. the analysis of **true** branch
   i. filters pre-condition
   ii. computes the post-condition for the assignment of $\mathrm{y} := \mathrm{x} - 3$
   iii. we get : $\{\mathrm{x} \mapsto [4, +\infty), \mathrm{y} \mapsto [1, +\infty)\}$
2. the analysis of **false** branch
   - we get : $\{\mathrm{x} \mapsto (-\infty, 7], \mathrm{y} \mapsto [0, +\infty)\}$
3. abstract join of these two abstract states
   - we get : $\{\mathrm{x} \mapsto \top, \mathrm{y} \mapsto [0, +\infty)\}$

Conditional Commands with a Relational Abstract Domain (1/1)

We have to use different algorithm:

- for the analysis of condition tests
- for the computation of abstract join

Analysis of conditional test with a relational domain :

- add several constraints to the abstract states

In general, it is more precise. Condition test that involve several variables are more precise. (more likely to be presented exactly)

- Consider the case of $\mathbf{x} \leq \mathbf{y}$

Abstract Interpretation of Loops (1/n)

Concrete Semantics of Loop

> $$[\![\mathtt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\left(\cup_{i\geq 0} \left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\right)$$

Note:

- Over-approximation of $[\![C]\!]_{\mathscr{P}}$ can be computed.
- Over-approximation of sequences of commands can be obtained by the over-approximation of each commands.

That is,

- Over-approximation of $[\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B$ can be computed

Abstract Interpretation of Loops (2/n)

Concrete Semantics of Loop

> $$[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\Big(\cup_{i\geq 0}\left([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\right)^i(M)\Big)$$

- $F = [\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B$
- $F^\sharp$: over-approximation of F

Goal:

- Over-approximation of the infinite union $\cup_{i\geq 0}F^i(M)$ with $F^\sharp$

Abstract Interpretation of Loops (3/n)

Example 3.16 (Analysis of programs with loops)

We will use these programs as a example.

Figure 3.9(a)

```
x := 0;
while (x >= 0) {
    x := x + 1;
}
```

Figure 3.9(b)

```
x := 0;
while (x <= 100) {
    if (x >= 50) {
        x := 10
    } else {
        x := x + 1
    }
}
```

Sequences of Concrete and Abstract Iterates (1/n)

Situation : a loop iterates at most n times. (n is a fixed integer value)

Then, the states they may generate at the loop head are :

- $M_n = \bigcup_{i=0}^{n} F^i(M)$

The sequences $(M_k)_{k\in\mathbb{N}}$ can be defined recursively as follows :

- $M_0 = M$
- $M_{k+1} = M_k \cup F(M_k)$

Then,

- over-approximation of M_n : can be easily done using $\sqcup^\sharp$ (, which is used in the previous chapter)

Sequences of Concrete and Abstract Iterates (2/n)

Indeed, let us assume:

- $M^\sharp$: an abstract element of the abstract domain
  - $M \subseteq \gamma(M^\sharp)$

We define the abstract iterates $(M_k^\sharp)_{k\in\mathbb{N}}$ as follows

- $M_0^\sharp = M^\sharp$
- $M_{k+1}^\sharp = M_k^\sharp \sqcup^\sharp F^\sharp(M_k^\sharp)$

Then we can prove by induction that

- for all integers n, $M_n \subseteq \gamma(M_n^\sharp)$

Proof of $\forall n$, $M_n \subseteq \gamma(M_n^\sharp)$

1. $n = 0$
    - It is obvious from assumption that $M_0 \subseteq \gamma(M_0^\sharp)$
2. $n = k$
    - we assume that $M_k \subseteq \gamma(M_k^\sharp)$
    - M_{k+1}
        - $= M_k \cup F(M_k)$
        - $\subseteq \gamma(M_k^\sharp) \cup F(\gamma(M_k^\sharp))$ ($\because M_k \subseteq \gamma(M_k^\sharp)$)
        - $\subseteq \gamma(M_k^\sharp) \cup \gamma(F^\sharp(M_k^\sharp))$ ($\because$ soundness of $F^\sharp$)
        - $\subseteq \gamma(M_k^\sharp \sqcup^\sharp F^\sharp(M_k^\sharp))$ ($\because$ soundness of $\sqcup^\sharp$)
        - $= \gamma(M_{k+1}^\sharp)$
    - $\therefore M_{k+1} \subseteq \gamma(M_{k+1}^\sharp)$

Sequences of Concrete and Abstract Iterates (3/n)

Example 3.17 (Abstract iterates)

Figure 3.9(a)

```
x := 0;
while (x >= 0) {
    x := x + 1;
}
```

Figure 3.9(b)

```
x := 0;
while (x <= 100) {
    if (x >= 50) {
        x := 10
    } else {
        x := x + 1
    }
}
```

Sequences of Concrete and Abstract Iterates (4/n)

Example 3.17 (Abstract iterates)

In the case of program (a):

- $M_0^\sharp = \{\mathrm{x} \mapsto [0, 0]\}$
- $M_1^\sharp = \{\mathrm{x} \mapsto [0, 1]\}$
- $M_2^\sharp = \{\mathrm{x} \mapsto [0, 2]\}$
- ...
- $M_n^\sharp = \{\mathrm{x} \mapsto [0, n]\}$
- ...

In the case of program (b):

- ...
- $M_{49}^\sharp = \{\mathrm{x} \mapsto [0, 49]\}$
- $M_{51}^\sharp = \{\mathrm{x} \mapsto [0, 50]\}$
- $M_{52}^\sharp = \{\mathrm{x} \mapsto [0, 50]\}$
- $M_{53}^\sharp = \{\mathrm{x} \mapsto [0, 50]\}$
- ...

Convergence of Iterates (1/3)

> $M^\sharp_{k+1} = M^\sharp_k \sqcup^\sharp F^\sharp(M^\sharp_k)$

We consider :

- the case of unbounded iteration
- the termination problem

Let us assume that :

- the abstract iteration stabilize at some rank n

Then,

- for all $k \geq n$, $\quad M^\sharp_k = M^\sharp_n \;$ and $\; M_k \subseteq \gamma(M^\sharp_n)$

Also,

- $M_{\text{loop}} \subseteq \gamma(M^\sharp_n) \quad$ where $\quad M_{\text{loop}} = \bigcup_{i \geq 0} M_i$

Convergence of Iterates (2/3)

Another interesting observation is that :

- $M_{\text{loop}} = \bigcup_{i \geq 0} F^i(M) = \bigcup_{i \geq 0} M_i \subseteq \gamma(M_n^\sharp)$

If the sequences of abstract iterates converges :

- its final value over-approximate *all* the concrete behaviors of **`while`**$(B)(C)$.

> If the sequences of abstract iterates converges

This can be observed by checking two consecutive iterates.

Convergence of Iterates (3/3)

Example 3.18 (Convergence of abstract iterates)

- In the case of program (a) :
    - the sequences of abstract iterates does not converge.
- In the case of program (b) :
    - the ranges of $\mathbf{x}$ stabilize but only after 51 iterations.

Neither of these are satisfactory.

- lack of termination
- hight number required to stabilize

We have to formalize the condition that ensures that

- the sequences of abstract iterates converges.

Convergence in Finite Height Lattices (1/4)

Assumption:

- $\sqsubseteq$ is such that
  - $M_a^\sharp \sqsubseteq M_b^\sharp \quad$ if and only if $\quad \gamma(M_a^\sharp) \subseteq \gamma(= M_b^\sharp)$ for all abstract states $M_a^\sharp, M_b^\sharp$

First case where convergence is ensured is when:

- $M_a^\sharp \sqsubset M_b^\sharp$

cannot hold infinitely many times.

This condition is realized when

- the abstract domain has ***finite height***, or
- the length of the chain below is bounded by some fixed value h (*height of the abstract domain*).
  - $M_0^\sharp \sqsubset M_1^\sharp \sqsubset \cdots \sqsubset M_k^\sharp$

Convergence in Finite Height Lattices (2/4)

For example, if the abstract domain has finite height h, the sequences

- $M_0^\sharp, M_1^\sharp, \cdots, M_h^\sharp, M_{h+1}^\sharp$

is increasing for $\sqsubseteq$, but cannot be strictly increasing.

So there exists a number $n (\leq h)$

- at which it becomes stable.
- which is bounded by the height of lattice.

Convergence in Finite Height Lattices (3/4)

- $M^\sharp_{\text{lim}}$: over-approximation of M_{loop}
- $M^\sharp_{\text{lim}}$ can be computed by the algorithm below :

Figure 3.10 (a)

- abs_iter$(F^\sharp, M^\sharp)$
  - $R \longleftarrow M^\sharp;$
  - repeat
    - $T \longleftarrow R;$
    - $R \longleftarrow R \sqcup^\sharp F^\sharp(R);$
  - until $R = T$
  - return $M^\sharp_{\text{lim}} = T;$

Convergence in Finite Height Lattices (4/4)

Example 3.19 (Convergence of abstract iterates in the signs abstract domain)

- domain : signs abstract domain
- program : same as example 3.16 and 3.17
- In the case of the program of (a), we obtain :
  - $M_0^\sharp = \{\mathrm{x} \mapsto [= 0]\}$
  - $M_1^\sharp = \{\mathrm{x} \mapsto [\geq 0]\}$
  - $M_2^\sharp = \{\mathrm{x} \mapsto [\leq 0]\}$
    - this analysis terminates after only two iterations
- In the case of the program of (b), we obtain the same result.

Widening Operators (1/7)

- We will use ***widening*** technique for iterates to converge quickly.
- Essentially, widening operator do:
    - over-approximate concrete unions
    - enforces termination of all sequences of iteration

Widening Operators (2/7)

Definition 3.11 (Widening operator)

- widening operator : ∇ such that
  i. for all abstract elements a_0 and a_1, $\quad \gamma(a_0) \cup \gamma(a_1) \subseteq \gamma(a_0 \nabla a_1)$
  ii. for all sequences $(a_n)_{n\in\mathbb{N}}$ of abstract elements, the sequences of $(a'_n)_{n\in\mathbb{N}}$ defined below is ultimately stationary (= eventually converge).
    - $a'_0 = a_0$
    - $a'_{n+1} = a'_n \nabla a_n$

Then we can turn the sequence of abstract iterates into a terminating sequence.

Widening Operators (3/7)

Theorem 3.5 (Abstract iterates with widening)

Let we assume:

- ∇ : widening operator over non-relational abstract domain $\mathbb{A}$
- $F^\sharp : \mathbb{A} \to \mathbb{A}$

Then, the algorithm shown in the next page terminates and returns $M^\sharp_{\mathrm{lim}}$.

Widening Operators (4/7)

Figure 3.10 (b)

- abs_iter$(F^\sharp, M^\sharp)$
  - $R \longleftarrow M^\sharp$;
  - repeat
    - $T \longleftarrow R$;
    - $R \longleftarrow R \triangledown F^\sharp(R)$;
  - until $R = T$
  - return $M^\sharp_{\text{lim}} = T$;

Widening Operators (5/7)

Theorem 3.5 (Abstract iterates with widening) (continued)

Let we assume:

- $F : \mathbb{M} \to \mathbb{M}$
  - continuous
  - $F \circ \gamma \subseteq \gamma \circ F^\sharp$ (in the sense of point-wise)

Then,

- $\bigcup_{i \geq 0} F^i(\gamma(M^\sharp)) \subseteq \gamma(M^\sharp_{\text{lim}})$
  - $M^\sharp_{\text{lim}}$ over-approximates the concrete semantics of the loop.

This theorem guarantees

- the termination of the loop analysis

Widening Operators (6/7)

Widening operator for the intervals domain would be like this:

- $[np] \nabla_{\mathscr{V}} [n, q] =$
  - $[n, p]$ if $p \geq q$
  - $[n, +\infty)$ if $p < q$

Widening Operators (7/7)

Example 3.20 (Widening operator for he abstract domain of intervals)

- program : same as example 3.16 and 3.17
- In both case, we obtain the following iteration sequence:
  - $M_0^\sharp = \{\mathrm{x} \mapsto [0, 0]\}$
  - $M_1^\sharp = \{\mathrm{x} \mapsto [0, +\infty)\}$
  - $M_2^\sharp = \{\mathrm{x} \mapsto [0, +\infty)\}$
- The convergence is now very fast, however
  - the result is coarse in the case of program (b),
    - this analysis doesn't converge.
- Some common techniques to obtain more precise result is in section 5.2

Analysis of Loops (1/1)

- semantics of the analysis of loop
  - $[\![\texttt{while}(B)\{C\}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = \mathscr{F}^\sharp_{\neg B}(\mathrm{abs\_iter}([\![C]\!]^\sharp_{\mathscr{P}} \circ \mathscr{F}^\sharp_B, M^\sharp))$

Analysis of Loops with a Relational Abstract Domain

- Almost same as with a non-relational domain
- Requires only an abstract join or widening operator specific to the abstraction being used

That is,

- In the case of linear equalities
  - widening is not necessary because its height of lattice is finite
- In the case of convex polyhedra and octagons
  - widening operator is required because its height of lattice is infinite.

Another View on the Analysis of Loops (1/n)

- concrete semantics of a loop statement
  - $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}\Big(\cup_{i\geq 0} \, ([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B)^i(M)\Big)$
    - $= \mathscr{F}_{\neg B}(M_{\mathrm{loop}})$

Let us consider the following equation:

- $M_{\mathrm{loop}} = \cup_{i\geq 0}([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B)^i(M)$
  - $= M \cup \Big(\bigcup_{i>0}([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B)^i(M)\Big)$
  - $= M \cup [\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B\Big(\bigcup_{i\geq 0}([\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B)^i(M)\Big)$
  - $= M \cup [\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B(M_{\mathrm{loop}})$

Another View on the Analysis of Loops (2/n)

Observation:

- M_{loop} is a ***fixpoint*** of a function $G : X \mapsto M \cup [\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B(X)$
- M_{loop} is a smallest set of states. M_{loop} is a ***least fixpoint*** of G

We let $\mathbf{lfp}\, G$ denote the least fixpoint of G.

Then, **concrete semantics** of a loop can be expressed like this

- $[\![\texttt{while}(B)\{C\}]\!]_{\mathscr{P}}(M) = \mathscr{F}_{\neg B}(\mathbf{lfp}\, G)$
  - where $G : X \mapsto M \cup [\![C]\!]_{\mathscr{P}} \circ \mathscr{F}_B(X)$

Another View on the Analysis of Loops (2/n)

- **abstract semantics** of a loop relies on the over-approximation of a concrete least fixpoint.
- When the abstract lattice has
  - *finite* height
    - we use abstract union
  - *infinite* height
    - we use widening operator
- We will see several improvements in section 5.2.

That's all 🙂

- $[\![n]\!]^\sharp(M^\sharp) = \phi_{\mathscr{V}}(n)$
- $[\![\mathrm{x}]\!]^\sharp(M^\sharp) = M^\sharp(\mathrm{x})$
- $[\![\mathtt{E}_0 \odot \mathtt{E}_1]\!]^\sharp(M^\sharp) = f^\sharp_\odot([\![\mathtt{E}_0]\!]^\sharp(M^\sharp), [\![\mathtt{E}_1]\!]^\sharp(M^\sharp))$
- $[\![\mathtt{C}]\!]^\sharp_{\mathscr{P}}(\perp) = \perp$
- $[\![\mathtt{skip}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = M^\sharp$
- $[\![\mathtt{C}_0; \mathtt{C}_1]\!]^\sharp_{\mathscr{P}}(M^\sharp) = [\![\mathtt{C}_0]\!]^\sharp_{\mathscr{P}}([\![\mathtt{C}_1]\!]^\sharp_{\mathscr{P}}(M^\sharp))$
- $[\![\mathrm{x} := \mathtt{E}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto [\![\mathtt{E}]\!]^\sharp(M^\sharp)]$
- $[\![\mathtt{input}(\mathrm{x})]\!]^\sharp_{\mathscr{P}}(M^\sharp) = M^\sharp[\mathrm{x} \mapsto \top_{\mathscr{V}}]$
- $[\![\mathtt{if}(B)\{C_0\}\mathtt{else}\{C_1\}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = [\![C_0]\!]^\sharp_{\mathscr{P}}(\mathscr{F}^\sharp_B(M^\sharp)) \sqcup^\sharp [\![C_1]\!]^\sharp_{\mathscr{P}}(\mathscr{F}^\sharp_{\neg B}(M^\sharp))$
- $[\![\mathtt{while}(B)\{C\}]\!]^\sharp_{\mathscr{P}}(M^\sharp) = \mathscr{F}^\sharp_{\neg B}(\mathrm{abs\_iter}([\![C]\!]^\sharp_{\mathscr{P}} \circ \mathscr{F}^\sharp_B, M^\sharp))$

Overview

- Semantics (3.1)
- Abstraction (3.2)
- Computable Abstract Semantics (3.3)
  - introduction
  - semantics of each commands
  - soundness
- Interpreter (3.4)

Soundness (1/2)

Theorem 3.6 (Soundness)

For all commands C and all abstract states $M^\sharp$, the computation of $\gamma(\llbracket C \rrbracket^\sharp_{\mathscr{P}}(M^\sharp))$ terminates and:

- $\llbracket C \rrbracket_{\mathscr{P}}(\gamma(M^\sharp)) \subseteq \gamma(\llbracket C \rrbracket^\sharp_{\mathscr{P}}(M^\sharp))$
  - Proof : by the induction over the syntax of commands.
    - For each kind of commands, we ensured that the definition of its semantics would lead to sound result.

Soundness (2/2)

We can also use best abstraction function α instead of γ.

- $\alpha([\![C]\!]_{\mathscr{P}}(M)) \sqsubseteq [\![C]\!]^{\sharp}_{\mathscr{P}}(\alpha(M))$

Analysis of the whole program (1/n)

For instance,

- program : C
- initial state : $\gamma(M^\sharp)$
- output state : $\gamma(\llbracket C \rrbracket^\sharp_{\mathscr{P}}(M^\sharp))$
- property of interest : M

Analysis of the whole program (2/n)

In general, if the inclusion does not hold, ***alarms*** will be called.

- alarms : says that the analysis tools failed to prove the property of interest
- triage :
  i. inspect the result of the analysis
  ii. decide whether the alarm is true or false

Note:

- The analysis function $[\![C]\!]^{\sharp}_{\mathscr{P}}$ is not monotone.
  - Therefore, replacing pre-condition $M^{\sharp}$ with more precise one does not ensure that the result is more precise.

Different Abstraction

What if we want to use another abstraction.

The analysis of

- expression
- input

is essentially non-relational abstraction and it has to be modified.

However, in general, overall structure of the analysis doesn't need to be modified.

Overview

- Semantics (3.1)
- Abstraction (3.2)
- Computable Abstract Semantics (3.3)
- Interpreter (3.4)

Interpreter (1/5)

General three steps to construct a static analysis:

1. fix the reference concrete semantics
2. select the abstraction
3. derive analysis algorithm

Interpreter (2/5)

1. Concrete Semantics

- $[\![C]\!]_{\mathscr{P}} : \wp(\mathbb{M}) \longrightarrow \wp(\mathbb{M})$
  - $\mathbb{M}$: set of memory states
  - $f_{\odot}$: operations for each operator in the language
  - $\mathscr{F}_{\mathtt{B}}$: filter functions
  - $\cup$: union
  - infinite set union, least fixpoint

Interpreter (3/5)

2. Abstraction

- $\mathbb{A} = (\mathbb{X} \longrightarrow \mathbb{A}_{\mathscr{V}})$
- $\gamma : \mathbb{A} \longrightarrow \wp(\mathbb{M})$

Note:

- Actual definition relies on
  - the value abstraction $\mathbb{A}_{\mathscr{V}}$
  - the concretization function $\gamma_{\mathscr{V}}$

Interpreter (4/5)

3. Abstract Semantics

- $[\![C]\!]^\sharp_{\mathscr{P}} : \mathbb{A} \longrightarrow \mathbb{A}$

Note:

- Actual definition relies on
  - $f^\sharp_\odot$: sound over-approximation of $f_\odot$
  - $\mathscr{F}^\sharp_{\mathtt{B}}$: abstract filter function (which is sound with respect to $\mathscr{F}_{\mathtt{B}}$)
  - $\sqcup^\sharp$: sound over-approximation of $\cup$
  - over-approximation of concrete fixpoint
    - based on a widening operator

Interpreter (5/5)

This division of the analysis design into independent steps is important

- for the construction of a static analysis
- when a static analysis needs to be improved (a static analysis is imprecise)

Common case a static analysis is imprecise:

- abstraction is coarse (step 2)
- algorithm return overly approximated result (step 3)
- concrete semantics is too coarse to express the properties of interest (step 1)