A General Static Analysis Framework Based on a Compositional Semantics

Material covered in chapter 3.3 of
Introduction to Static Analysis: an Abstract Interpretation Perspective

Purpose of this lecture

So far, we have learned:

1. **programming language semantics**:
   i.e., how to give meaning to programs
2. **abstraction of semantic behaviors**:
   i.e., how to compare two different meanings
   intuitively, concrete is more expressive, abstract simpler to represent

The next step is to compute an abstraction of the semantics of programs

This computation shall be guided by the semantics and the abstraction...

Content of the lecture:

- abstract interpretation of all commands in the simple language defined previously and **construction of a static analyzer**
- definition of abstract opreations
- design of a terminating analysis for loops

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Language syntax

Assumptions: set of values n, finite set of variables $\mathbb{X}$

E	$::=$		scalar expressions
	$\mid$	n	scalar constant $n \in \mathbb{V}$
	$\mid$	$\mathtt{x}$	variable $\mathtt{x} \in \mathbb{X}$
	$\mid$	$E \odot E$	binary operation
B	$::=$		Boolean expressions
	$\mid$	$\mathtt{x} \lessdot n$	comparison of a variable with a constant
C	$::=$		commands
	$\mid$	$\mathtt{skip}$	command that “does nothing"
	$\mid$	$C;\ C$	sequence of commands
	$\mid$	$\mathtt{x} := E$	assignment command
	$\mid$	$\mathtt{input}(\mathtt{x})$	command reading of a value
	$\mid$	$\mathtt{if}(B)\{C\}\mathtt{else}\{C\}$	conditional command
	$\mid$	$\mathtt{while}(B)\{C\}$	loop command
P	$::=$	C	program

Concrete semantics definition

In this lecture, we use a **compositional semantics**.

We first recall the **main definitions**:

- set of base values: $\mathbb{V}$
- set of variables: $\mathbb{X}$ (fixed and finite)
- set of memory states: $\mathbb{M} = \mathbb{X} \longrightarrow \mathbb{V}$

Then, the **semantics** are defined by **induction over the syntax**, and have the **following types**:

- semantics of an expression E: $[\![E]\!] : \mathbb{M} \longrightarrow \mathbb{V}$
  (maps a memory state into a value)
- semantics of a condition B: $[\![B]\!] : \mathbb{M} \longrightarrow \mathbb{B}$
- semantics of a command C: $[\![C]\!] : \wp(\mathbb{M}) \longrightarrow \wp(\mathbb{M})$
  (maps a pre-condition set of memory states into a post-condition set)

Concrete semantics

Definition by induction over the syntax (from previous lectures):

$$
\begin{array}{rcl}
[\![n]\!](m) & = & n \\
[\![\mathtt{x}]\!](m) & = & m(\mathtt{x}) \\
[\![\mathtt{E}_0 \odot \mathtt{E}_1]\!](m) & = & f_{\odot}([\![\mathtt{E}_0]\!](m), [\![\mathtt{E}_1]\!](m)) \\
[\![\mathtt{x} \otimes n]\!](m) & = & f_{\otimes}(m(\mathtt{x}), n) \\
[\![\mathtt{skip}]\!]_{\mathcal{P}}(M) & = & M \\
[\![\mathtt{C}_0;\ \mathtt{C}_1]\!]_{\mathcal{P}}(M) & = & [\![\mathtt{C}_1]\!]_{\mathcal{P}}([\![\mathtt{C}_0]\!]_{\mathcal{P}}(M)) \\
[\![\mathtt{x} := \mathtt{E}]\!]_{\mathcal{P}}(M) & = & \{ m[\mathtt{x} \mapsto [\![\mathtt{E}]\!](m)] \mid m \in M \} \\
[\![\mathtt{input}(\mathtt{x})]\!]_{\mathcal{P}}(M) & = & \{ m[\mathtt{x} \mapsto n] \mid m \in M, n \in \mathbb{V} \} \\
[\![\mathtt{if}(\mathtt{B})\{\mathtt{C}_0\}\mathtt{else}\{\mathtt{C}_1\}]\!]_{\mathcal{P}}(M) & = & [\![\mathtt{C}_0]\!]_{\mathcal{P}}(\mathcal{F}_{\mathtt{B}}(M)) \cup [\![\mathtt{C}_1]\!]_{\mathcal{P}}(\mathcal{F}_{\neg \mathtt{B}}(M)) \\
[\![\mathtt{while}(\mathtt{B})\{\mathtt{C}\}]\!]_{\mathcal{P}}(M) & = & \mathcal{F}_{\neg \mathtt{B}} \left(\bigcup_{i \geq 0} ([\![\mathtt{C}]\!]_{\mathcal{P}} \circ \mathcal{F}_{\mathtt{B}})^i (M) \right)
\end{array}
$$

where $f_{\odot}, f_{\otimes}$ define the evaluation of basic operations.

Value abstraction

In this course, we assume a rather simple abstraction, namely a **non-relational abstraction**, to fully show the definition of an abstract interpreter.

A value abstraction is defined by

- a lattice of abstract values $(\mathbb{A}_{\mathcal{V}}, \sqsubseteq_{\mathcal{V}})$
- concretization function $\gamma_{\mathcal{V}} : \mathbb{A}_{\mathcal{V}} \to \wp(\mathbb{V})$

We may also require a best abstraction $\alpha_{\mathcal{V}}$ defining a Galois connection, but it is not strictly necessary for building up an analysis.

Examples:

- constants
- intervals
- ...

Non-relational abstraction

The construction of **non-relational abstraction** is **based** on the **value abstraction**, and lifts it to functions over variables:

- 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 $\gamma_{\mathcal{N}}$

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

Note: when one variable is mapped to $\perp$, the abstract states describes $\emptyset$ and should be reduced to $\perp_{\mathcal{N}} = \mathrm{x} \mapsto \perp$.

Example: $\mathrm{x} \mapsto [1, 8]; \mathrm{y} \mapsto [2, 3]$ describes $m_0 = (\mathrm{x} \mapsto 1; \mathrm{y} \mapsto 2)$, $m_1 = (\mathrm{x} \mapsto 1; \mathrm{y} \mapsto 3), \ldots$

Abstract interpretation principle

We have defined:

- $[\![\mathtt{C}]\!] : \wp(\mathbb{M}) \longrightarrow \wp(\mathbb{M})$
- $\gamma_{\mathcal{N}} : \mathbb{A}_{\mathcal{N}} \longrightarrow \wp(\mathbb{M})$

Thus, $[\![\mathtt{C}]\!] \circ \gamma_{\mathcal{N}}(M_0^\sharp)$ describes the states that are reachable after running $\mathtt{C}$ from a state described by $M^\sharp$.

Let us automate the search for **an over approximation** of this set, that we could describe **by an abstract state** $M_1^\sharp$:

Abstract interpretation

An **abstract interpretation** of $[\![\mathtt{C}]\!]$ is defined by a function $[\![\mathtt{C}]\!]_{\mathcal{P}}^\sharp : \mathbb{A}_{\mathcal{N}} \longrightarrow \mathbb{A}_{\mathcal{N}}$ such that we have, for the pointwise ordering:

$$[\![\mathtt{C}]\!] \circ \gamma_{\mathcal{N}} \subseteq \gamma_{\mathcal{N}} \circ [\![\mathtt{C}]\!]_{\mathcal{P}}^\sharp$$

Then, we can simply let $M_1^\sharp = [\![\mathtt{C}]\!]_{\mathcal{P}}^\sharp(M_0^\sharp)$.

Thus, we now simply need to define $[\![\mathtt{C}]\!]_{\mathcal{P}}^\sharp$ for all $\mathtt{C}$.

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Principle

How to do **abstract interpretation of basic commands**?

Trivial case: **skip commands**

- **concrete semantics**: $[\![\mathtt{skip}]\!]_{\mathcal{P}}(M) = M$
- obvious choice for the **abstract semantics**: $[\![\mathtt{skip}]\!]^{\sharp}_{\mathcal{P}}(M^{\sharp}) = M^{\sharp}$

Let us look at a more interesting case, namely **assignments**

We have seen

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

The evaluation steps are:

1. evaluate expression E into a value v
2. update variable $\mathtt{x}$ with the result v

To follow this scheme in the abstract, we first need to **analyze expressions** in a given abstract state.

Analysis of expressions

The concrete semantics of expression was defined **by induction over the syntax**, so we follow this order:

- case of a **constant expression** n:
  we need a sound approximation $\phi_{\mathcal{V}}(n)$ of n in the abstract;
  if there exists a best abstraction function α, $\alpha(\{n\})$ works!
  then, $[\![n]\!]^\sharp(M^\sharp) = \phi_{\mathcal{V}}(n)$
- case of a **variable lookup** $\mathtt{x}$:
  we can simply read $\mathtt{x}$ in $M^\sharp$
  thus, $[\![\mathtt{x}]\!]^\sharp(M^\sharp) = M^\sharp(\mathtt{x})$
- case of a **binary operation expression** $E_0 \odot E_1$:
  we need a sound approximation $f_\odot^\sharp$ of $f_\odot$, i.e., that should satisfy

$$\forall n_0^\sharp, n_1^\sharp \in \mathbb{A}_{\mathcal{V}},\ f_\odot(\gamma_{\mathcal{V}}(n_0), \gamma_{\mathcal{V}}(n_1)) \subseteq \gamma_{\mathcal{V}}(f_\odot^\sharp(n_0^\sharp, n_1^\sharp))$$

  then, $[\![E_0 \odot E_1]\!]^\sharp(M^\sharp) = f_\odot^\sharp([\![E_0]\!]^\sharp(M^\sharp), [\![E_1]\!]^\sharp(M^\sharp))$

Soundness of the analysis of expressions

General principle when **designing and proving** abstract interpreters:

- identify the **soundness property** of each operation
- generally prove it **compositionally** and **by induction over the syntax**

Case of expressions:

Theorem: soundness of the analysis of expressions

The analysis of expressions is **sound** in the sense that, for all expression E

$$\forall M^\sharp \in \mathbb{A}_{\mathcal{N}},\ \forall m \in \gamma_{\mathcal{N}}(M^\sharp),\ [\![E]\!](m) \in \gamma_{\mathcal{V}}([\![E]\!]^\sharp(M^\sharp))$$

- intuition: the abstract semantics of epression computes an over-approximation of the set of concrete values it may produce
- the proof indeed proceeds by induction, just like the definition

Analysis of assignment commands and soundness

Let us put it all together. The analysis of an assignment command should:

1. analyze expression E into a abstract value;
2. update variable $\mathtt{x}$ with that abstract value in the non-relational domain.

We get:

$$[\![\mathtt{x} := E]\!]^{\sharp}_{\mathcal{P}}(M^{\sharp}) = M^{\sharp}[\mathtt{x} \mapsto [\![E]\!]^{\sharp}(M^{\sharp})]$$

The case of the **random input command** is similar:

$$[\![\mathtt{input}(\mathtt{x})]\!]^{\sharp}_{\mathcal{P}}(M^{\sharp}) = M^{\sharp}[\mathtt{x} \mapsto \top_{\mathcal{V}}]$$

Example

Let us assume that:

- there are only two variables x, y
- E is $\mathrm{x} * \mathrm{x} - \mathrm{y} * \mathrm{y}$ and the command is $\mathrm{x} = \mathrm{x} * \mathrm{x} - \mathrm{y} * \mathrm{y}$
- in the interval domain, $M^\sharp(\mathrm{x}) = [3, 4]$ and $M^\sharp(\mathrm{y}) = [-1, 0]$

Then:

$$\begin{array}{rcl} [\![E]\!]^\sharp(M^\sharp) & = & [\![\mathrm{x} * \mathrm{x}]\!]^\sharp(M^\sharp) -^\sharp [\![\mathrm{y} * \mathrm{y}]\!]^\sharp(M^\sharp) \\ & = & [\![\mathrm{x}]\!]^\sharp(M^\sharp) *^\sharp [\![\mathrm{x}]\!]^\sharp(M^\sharp) -^\sharp [\![\mathrm{y}]\!]^\sharp(M^\sharp) *^\sharp [\![\mathrm{y}]\!]^\sharp(M^\sharp) \\ & = & [3, 4] *^\sharp [3, 4] -^\sharp [-1, 0] *^\sharp [-1, 0] \\ & = & [8, 16] \end{array}$$

and:

$$[\![\mathrm{x} := \mathrm{x} * \mathrm{x} - \mathrm{y} * \mathrm{y}]\!]_{\mathcal{P}}^\sharp(M^\sharp) = \{\mathrm{x} \mapsto [8, 16], \mathrm{y} \mapsto [-1, 0]\}$$

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Analysis of sequences

Next command: **sequence command** $C_0;\ C_1$

- important constraint over the analysis design:
  it is better to let the analysis consist of a **composition** of **local** operations
- let us make an additional assumption:
  we already know how to analyze C_0 and C_1,
  i.e., $[\![C_0]\!]^\sharp_{\mathcal{P}}$ and $[\![C_1]\!]^\sharp_{\mathcal{P}}$ are well defined and sound.

Then, given an abstract state $M^\sharp$:

$$\begin{array}{rcll} [\![C_0;\ C_1]\!](\gamma_{\mathcal{N}}(M^\sharp)) & = & [\![C_1]\!] \circ [\![C_0]\!] \circ \gamma_{\mathcal{N}}(M^\sharp) & \\ & \sqsubseteq & [\![C_1]\!] \circ \gamma_{\mathcal{N}} \circ [\![C_0]\!]^\sharp_{\mathcal{P}}(M^\sharp) & (\text{monotonicity of } [\![C_1]\!] \\ & & & \text{soundness of } [\![C_0]\!]^\sharp_{\mathcal{P}}) \\ & \sqsubseteq & \gamma_{\mathcal{N}} \circ [\![C_1]\!]^\sharp_{\mathcal{P}} \circ [\![C_0]\!]^\sharp_{\mathcal{P}}(M^\sharp) & (\text{soundness of } [\![C_1]\!]^\sharp_{\mathcal{P}}) \end{array}$$

This suggests to simply let $[\![C_0;\ C_1]\!]^\sharp_{\mathcal{P}} = [\![C_1]\!]^\sharp_{\mathcal{P}} \circ [\![C_0]\!]^\sharp_{\mathcal{P}}$

Soundness

We now have a **clearer view into the design of $[\![.]\!]^{\sharp}_{\mathcal{P}}$**:

- definition **by induction** over the syntax of commands
- proof **also by induction**

Basic commands (skip, assignment, input) have been discussed.

Two major remaining cases, that require a lot more work:

- conditions
- loops

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Semantics of the condition command and abstraction

We consider **condition** $\texttt{if}(B)\{C_0\}\texttt{else}\{C_1\}$:

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

We need to analyze:

1. the **condition functions** $\mathcal{F}_B$ and $\mathcal{F}_{\neg B}$
2. the body of the **blocks** C_0, C_1
   already done: we are building $[\![.]\!]^{\sharp}_{\mathcal{P}}$ by induction over the syntax
3. the **union** $\cup$

To analyze conditions, we need to consider $\mathcal{F}_{.}$ and $\cup$ and compose their abstractions.

Abstract condition operator definition

The analysis of $\mathcal{F}_{.}$ depends on the non-relational domain. We defer it to some specific operator:

Definition

Given a condition B, an **abstract condition operator** is a function $\mathcal{F}_B^\sharp : \mathbb{A}_\mathcal{N} \longrightarrow \mathbb{A}_\mathcal{N}$ that is sound in the following sense:

$$\forall M^\sharp \in \mathbb{A}_\mathcal{N},\ \mathcal{F}_B \circ \gamma_\mathcal{N}(M^\sharp) \subseteq \gamma_\mathcal{N} \circ \mathcal{F}_B^\sharp(M^\sharp)$$

- intution: $\mathcal{F}_B^\sharp$ inputs an abstract value that describes a set of constraints and adds one more constrait corresponding to B
- the precise definition depends on $\mathbb{A}_\mathcal{V}$

A condition operator for intervals

We assume $\mathbb{A}_{\mathcal{V}}$ is **the domain of intervals** and we consider a few cases.

We assume that $M^\sharp(\mathtt{x}) = [a, b]$. Then:

$$\mathcal{F}_{\mathtt{x} \leq n}(M^\sharp) \quad = \quad \begin{cases} \bot_{\mathcal{N}} \quad (= \lambda x \cdot \bot) & \text{if } n < a \\ M^\sharp[\mathtt{x} \mapsto [a, n]] & \text{if } a \leq n \leq b \\ M^\sharp & \text{if } b \leq n \end{cases}$$

Remarks:

- when the condition is not satisfiable, **reduction to $\bot$** should be performed
- more complex expressions can be considered as well
- returning $M^\sharp$ is always a sound behavior

Abstract union

The second step is to construct **a counterpart for the concrete union**, namely **a binary operator $\sqcup^\sharp$ over $\mathbb{A}_\mathcal{N}$ such that**:

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

This is done in two steps:

1. **definition of a sound abstract union $\sqcup_\mathcal{V}^\sharp$ for the value abstract domain**; for example, for intervals:

$$[a_0, b_0] \sqcup_\mathcal{V}^\sharp [a_1, b_1] = [\min(a_0, a_1), \max(b_0, b_1)]$$

2. **the pointwise extension**:

$$\forall \mathrm{x},\ (M_0^\sharp \sqcup^\sharp M_1^\sharp)(\mathrm{x}) = M_0^\sharp(\mathrm{x}) \sqcup_\mathcal{V}^\sharp M_1^\sharp(\mathrm{x})$$

Analysis of conditions and soundness

We recall the **concrete semantics**:

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

The **abstract semantics** follows the same steps, and has a very similar form:

$$[\![\mathtt{if}(B)\{C_0\}\mathtt{else}\{C_1\}]\!]^{\sharp}_{\mathcal{P}}(M^{\sharp}) = [\![C_0]\!]^{\sharp}_{\mathcal{P}}(\mathcal{F}^{\sharp}_B(M^{\sharp})) \sqcup^{\sharp} [\![C_1]\!]^{\sharp}_{\mathcal{P}}(\mathcal{F}^{\sharp}_{\neg B}(M^{\sharp}))$$

- soundness: by composing the soundness statements of $\mathcal{F}^{\sharp}_{\cdot}$, of $\sqcup^{\sharp}$, and of $[\![.]\!]^{\sharp}_{\mathcal{P}}$ (which is being defined by induction)
- more generally:
  the abstraction of function composition boils down **to the compostion of abstractions**

Example

We consider a **basic absolute value** routine:

$$\begin{array}{l}\texttt{if}(\mathtt{x} \leq 0)\{\\ \qquad \mathtt{y} := -\mathtt{x}\\ \}\texttt{else}\{\\ \qquad \mathtt{y} := \mathtt{x}\\ \}\end{array}$$

Assuming **abstract pre-condition** $M^\sharp = \{\mathtt{x} \mapsto [-8, 3], \mathtt{y} \mapsto [-123, 234]\}$:

- $\mathcal{F}^\sharp_{\mathtt{x}\leq 0}(M^\sharp) = \{\mathtt{x} \mapsto [-8, 0], \mathtt{y} \mapsto [-123, 234]\}$
- $\mathcal{F}^\sharp_{\neg(\mathtt{x}\leq 0)}(M^\sharp) = \{\mathtt{x} \mapsto [1, 3], \mathtt{y} \mapsto [-123, 234]\}$
- the **abstract post-condition** is:

$$\{\mathtt{x} \mapsto [-8, 3], \mathtt{y} \mapsto [-8, 8]\}$$

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Semantics of loops and abstraction attempt

Concrete semantics of a loop command:

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

We have **already defined** the following abstract operations:

- abstract condition tests:
  $\mathcal{F}^\sharp_B$ and $\mathcal{F}^\sharp_{\neg B}$ over-approximate $\mathcal{F}_B$ and $\mathcal{F}_{\neg B}$
- loop body analysis:
  by induction hypothesis, $[\![C]\!]^\sharp_{\mathcal{P}}$ over-approximates $[\![C]\!]_{\mathcal{P}}$
- abstract union: $\sqcup^\sharp$ over-approximates $\cup$

Can we simply compose these elements ?

It is **not trivial** as the **concrete semantics definition is infinite**

Towards an iterative analysis method

First, we rewrite the concrete semantics; for a given M, we let:

- $M_0 = M$
- $M_{n+1} = M_n \cup [\![\mathcal{C}]\!]_{\mathcal{P}} \circ \mathcal{F}_{B}(M_n$

Then, since $[\![\mathcal{C}]\!]_{\mathcal{P}}$ and $\cup$ commute:

$$\bigcup_{i \geq 0} ([\![\mathcal{C}]\!]_{\mathcal{P}} \circ \mathcal{F}_{B})^{i}(M) = \bigcup_{i \geq 0} M_i$$

Moreover, $M_0 \subseteq M_1 \subseteq \ldots$ define a chain.

This suggests to start from $M^\sharp$, let $F^\sharp = [\![\mathcal{C}]\!]^\sharp_{\mathcal{P}} \circ \mathcal{F}^\sharp_{B}$, and compute:

$$\begin{array}{rcl} M^\sharp_0 & = & M^\sharp \\ M^\sharp_{k+1} & = & M^\sharp_k \sqcup^\sharp F^\sharp(M^\sharp_k) \end{array}$$

Last we should return $\mathcal{F}^\sharp_{\neg B}(M^\sharp_\infty)$ where $M^\sharp_\infty$ is over-approximating all the iterates... But **how to compute this over-approximation** ?

Termination

We need **conditions under which an over-approximation can be found**. For instance,

Termination under finite chain condition

If $\mathbb{A}_{\mathcal{N}}$ **has no infinite chain** and $\sqsubseteq_{\mathcal{N}}$ is equivalent to $\subseteq$ up to $\gamma_{\mathcal{N}}$, then the sequence $M_0^\sharp, M_1^\sharp, \ldots, M_n^\sharp, \ldots$ eventually stabilizes, i.e., there exists a rank N such that for any $k \geq N$, we have $M_N^\sharp = M_k^\sharp$.
We may let:

$$[\![\mathtt{while}(B)\{C\}]\!]_{\mathcal{P}}^\sharp(M^\sharp) = \mathcal{F}_{\neg B}^\sharp(M_N^\sharp)$$

```
abs_iter(F♯, M♯)
        R ← M♯;
        repeat
            T ← R;
            R ← R ⊔♯ F♯(R);
        until R = T
        return M♯_lim = T;
```

- **Soundness**: abstract iterates over-approximate concrete iterates
- **Termination**: by chain condition

Widening

In many interesting cases, the **finite chain condition is violated**.
Example: with the lattice of intervals

Solution: **accelerate convergence using a novel iteration technique**

- why non termination with infinite chains ?
  because abstract union may yield infinite sequences
  $\mathrm{R}_0, \mathrm{R}_1 = \mathrm{R}_0 \sqcup^\sharp F^\sharp(\mathrm{R}_0), \mathrm{R}_2 = \mathrm{R}_1 \sqcup^\sharp F^\sharp(\mathrm{R}_1), \mathrm{R}_3 = \mathrm{R}_2 \sqcup^\sharp F^\sharp(\mathrm{R}_2), \ldots$
- to guarantee termination, we need to **generalize faster sequences of abstract properties**

Definition: widening operator

A **widening operator** is a binary operator ∇ such that,

1. for all abstract elements a_0, a_1, we have $\gamma(a_0) \cup \gamma(a_1) \subseteq \gamma(a_0 \nabla a_1)$
2. for all sequences $(a_n)_{n\in\mathbb{N}}$ of abstract elements, the sequence $(a'_n)_{n\in\mathbb{N}}$ defined by $a'_0 = a_0$ and $a'_{n+1} = a'_n \nabla a_n$ is stationary.

A widening for intervals

How to construct widening operators ?

- in many abstract domains, abstract elements stand for **finite conjunctions of constraints**
- **preserving only stable constraints** gives a general way to achieve termination, while guaranteeing soundness:
  at each iteration, either the number of constraints goes down, or the limit is reached

Application for intervals:

Widening for intervals with the same left bound

$$[n, p] \nabla_{\mathcal{V}} [n, q] = \begin{cases} [n, p] & \text{if } p \geq q \\ [n, +\infty) & \text{if } p < q \end{cases}$$

- $[0, 24] \nabla_{\mathcal{V}} [0, 18] = [0, 24]$
- $[0, 24] \nabla_{\mathcal{V}} [0, 24] = [0, +\infty[$
- the case of the left bound is symmetric

Abstract semantics for loops and soundness

Novel iterator, using **widening**:

$\texttt{abs\_iter}(F^\sharp, M^\sharp) ::=$
$\quad \texttt{R} \leftarrow M^\sharp;$
$\quad$ repeat
$\quad\quad \texttt{T} \leftarrow \texttt{R};$
$\quad\quad \texttt{R} \leftarrow \texttt{R} \triangledown F^\sharp(\texttt{R});$
$\quad$ until $\texttt{R} = \texttt{T}$
$\quad$ return $M^\sharp_{\lim} = \texttt{T};$

- **Soundness**: abstract iterates over-approximate concrete iterates
- **Termination**: by the definition of widening

Abstract semantics for a loop:

- let $F^\sharp = [\![C]\!]^\sharp_{\mathcal{P}} \circ \mathcal{F}^\sharp_{B}$
- compute

$$[\![\texttt{while}(B)\{C\}]\!]^\sharp_{\mathcal{P}}(M^\sharp) = \mathcal{F}^\sharp_{\neg B}(\texttt{abs\_iter}(F^\sharp, M^\sharp))$$

Example

A basic example of widening iteration:

A **loop with one counter**:

$$
\begin{array}{l}
\mathtt{x} = 0; \\
\mathtt{while}(\mathtt{x} < 1000)\{ \\
\quad \mathtt{x} := \mathtt{x} + 1; \\
\}
\end{array}
$$

Abstract iterates and ranges inferred for $\mathtt{x}$:

1. $[0, 0] \; \nabla_{\mathcal{V}} \; [1, 1] = [0, +\infty[$
2. $[0, +\infty[\nabla_{\mathcal{V}} \; [1, +\infty[= [0, +\infty[$

Observations:

- abstract intepretation **terminates after only two** iterations over the main loop
- the range for $\mathtt{x}$ at loop exit is **imprecise**: $[1000, +\infty[$

Refining widening iteration

There exist may ways to reduce the imprecision induced by widening.

Unrolling the first iterations:

- principle: use $\sqcup^\sharp_\mathcal{V}$ instead of $\triangledown_\mathcal{V}$ for the first few abstract iterations
- application: have a chance to more precisely capture behaviors specific to the first iterations of the loop

Computing additional iterations after convergence:

- principle: after a post-fixpoint is reached, compute additional iterations of the abstract semantics of the loop body
- application: refine the output of widening

These approaches (and others) are detailed in chapter 5.

Outline

1. Basic assumptions
2. Static analysis of assignment commands
3. Static analysis of sequences of instructions
4. Static analysis of conditions
5. Static analysis of sequences of loops
6. Conclusion

Abstract semantics

The whole definition of the analysis:

$$
\begin{array}{rcl}
[\![n]\!]^\sharp(M^\sharp) & = & \phi_{\mathcal{V}}(n) \\
[\![\mathtt{x}]\!]^\sharp(M^\sharp) & = & M^\sharp(\mathtt{x}) \\
[\![E_0 \odot E_1]\!]^\sharp(M^\sharp) & = & f^\sharp_\odot([\![E_0]\!]^\sharp(M^\sharp), [\![E_1]\!]^\sharp(M^\sharp)) \\
[\![C]\!]^\sharp_{\mathcal{P}}(\bot) & = & \bot \\
[\![\mathtt{skip}]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & M^\sharp \\
[\![C_0; C_1]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & [\![C_1]\!]^\sharp_{\mathcal{P}}([\![C_0]\!]^\sharp_{\mathcal{P}}(M^\sharp)) \\
[\![\mathtt{x} := E]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & M^\sharp[\mathtt{x} \mapsto [\![E]\!]^\sharp(M^\sharp)] \\
[\![\mathtt{input}(\mathtt{x})]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & M^\sharp[\mathtt{x} \mapsto \top_{\mathcal{V}}] \\
[\![\mathtt{if}(B)\{C_0\}\mathtt{else}\{C_1\}]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & [\![C_0]\!]^\sharp_{\mathcal{P}}(\mathcal{F}^\sharp_B(M^\sharp)) \sqcup^\sharp [\![C_1]\!]^\sharp_{\mathcal{P}}(\mathcal{F}^\sharp_{\neg B}(M^\sharp)) \\
[\![\mathtt{while}(B)\{C\}]\!]^\sharp_{\mathcal{P}}(M^\sharp) & = & \mathcal{F}^\sharp_{\neg B}(\mathtt{abs\_iter}([\![C]\!]^\sharp_{\mathcal{P}} \circ \mathcal{F}^\sharp_B, M^\sharp))
\end{array}
$$

Summary

Basic principles of **static analysis by abstrct interpretation**:

1. follow the **structure** of the concrete semantics
2. seek for an **over-approximation** of each operation in the concrete semantics
   in some cases, the best approximation may not be computable or too expensive
3. substitute union with **widening** to enforce termation of abstract iterates

The **proof of soundness** also follows the structure of the semantics!