Static Analysis: an Abstract Interpretation Perspective

Kwangkeun Yi

Seoul National University

5/2019@The 9th SSFT

This lecture is based on the following forthcoming book

Static Analysis: an Abstract Interpretation Perspective,
Yi and Rival, MIT Press

Outline

Our Interest

How to **verify specific properties about program executions before execution**:

- absence of run-time errors i.e., no crashes
- preservation of invariants

Verification

Make sure that $[\![P]\!] \subseteq \mathcal{S}$ where

- **the semantics** $[\![P]\!]$ = the set of all behaviors of P
- **the specification** $\mathcal{S}$ = the set of acceptable behaviors

Semantics $[\![P]\!]$ and Semantic Properties $\mathcal{S}$

Semantics $[\![P]\!]$:

- compositional style ("denotational")
  - $[\![AB]\!] = \cdots [\![A]\!] \cdots [\![B]\!] \cdots$
- transitional style ("operational")
  - $[\![AB]\!] = \{s_0 \hookrightarrow s_1 \hookrightarrow \cdots, \cdots\}$

Semantic properties $\mathcal{S}$:

- safety
  - some behavior observable in *finite* time will never occur.
- liveness
  - some behavior observable after *infinite* time will never occur.

Safety Properties

Some behavior observable in *finite* time will never occur.

Examples:

- absence of crashing error
  e.g., no uncaught exceptions in ML, no memory errors in C
- preservation of a general invariant
  e.g., some data structure should never get broken
- assertion on variable values
  e.g., the values of a variable always in a given range

Liveness Properties

Some behavior observable after *infinite* time will never occur.

Examples:

- no unbounded repetition of a given behavior
- no starvation
- no non-termination

Soundness and Completeness

"Analysis is sound." "Analysis is complete."

- **Soundness**: analysis(P) = yes $\Longrightarrow P$ satisfies the specification
- **Completeness**: analysis(P) = yes $\Longleftarrow P$ satisfies the specification

Spectrum of Program Analysis Techniques

- testing
- machine-assisted proving
- finite-state model checking
- **conservative static analysis**
- bug-finding

Testing

Approach

1. **Consider finitely many, finite executions**
2. For each of them, **check whether it violates the specification**

- If the finite executions find no bug, then accept.
- **Unsound**: can accept programs that violate the specification
- **Complete**: does not reject programs that satisfy the specification

Machine-Assisted Proving

Approach

1. Use a **specific language** to **formalize verification goals**
2. **Manually supply proof arguments**
3. Let the proofs be **automatically verified**

- tools: Coq, Isabelle/HOL, PVS, ...
- **Applications**: CompCert (certified compiler), seL4 (secure micro-kernel), ...
- **Not automatic**: key proof arguments need to be found by users
- **Sound**, if the formalization is correct
- **Quasi-complete** (only limited by the expressiveness of the logics)

Finite-State Model Checking

Approach

1. Focus on **finite state models** of programs
2. Perform **exhaustive exploration** of program states

- **Automatic**
- **Sound** or **complete**, **only with respect to the finite models**
- But, software has $\sim \infty$ states: need finite approximation or non-termination

Conservative Static Analysis

Principle

1. Perform **automatic verification, yet which may fail**
2. Compute a **conservative approximation of the program semantics**

- **Either sound or complete, not both**
- **Sound & incomplete** static analysis is common:
  - ML type systems, Astrée, Sparrow, Facebook Infer, ...
  - optimizing compilers relies on it
- **Automatic**
- Incompleteness: **may reject safe programs** or **may raise false alarms**
- Analysis algorithms **reason over program semantics**

Bug Finding

Approach

Automatic, **unsound** and **incomplete** algorithms

- Coverity, CodeSonar, SparrowFasoo, ...
- **Automatic** and **generally fast**
- **No mathematical guarantee about the results**
  - may reject a correct program, and accept an incorrect one
  - may raise false alarm and fail to report true violations
- Used to increase software quality without any guarantee

High-level Comparison

	automatic	sound	complete
testing	yes	no	yes
machine-assisted proving	no	yes	yes/no
finite-state model checking	yes	yes	yes/no
conservative static analysis	yes	yes	no
bug-finding	yes	no	no

Focus of This Lecture: Conservative Static Analysis

A general technique, for any programming language $\mathbb{L}$ and safety property $\mathcal{S}$, that

- **checks**, for input program P in $\mathbb{L}$, if $[\![P]\!] \subseteq \mathcal{S}$,
- **automatic** (algorithm)
- **finite** (terminating)
- **sound** (guarantee)
- **malleable** for arbitrary precision

A forthcoming framework

Will guide us how to design such static analysis.

Problem: How to Finitely Compute $[\![P]\!]$ Beforehand

- Finite & exact computation $\mathrm{Exact}(P)$ of $[\![P]\!]$ is **impossible, in general**.

  For a Turing-complete language $\mathbb{L}$,
  $\nexists$algorithm Exact : $\mathrm{Exact}(P) = [\![P]\!]$ for all P in $\mathbb{L}$.

- Otherwise, we can solve the Halting Problem.
  - Given P, see if $\mathrm{Exact}(P; 1/0)$ has divide-by-zero.

Answers: Conservative Static Analysis

Technique for **finite sound estimation** $[\![P]\!]^\sharp$ of $[\![P]\!]$

- "finite", hence
  - automatic (algorithm) &
  - static (without executing P)
- "sound"
  - over-approximation of $[\![P]\!]$

Hence, ushers us to sound anaysis:

$$(\text{analysis}(P) = \text{check } [\![P]\!]^\sharp \subseteq \mathcal{S}) \Longrightarrow (P \text{ satisfies property } \mathcal{S})$$

Need Formal Frameworks of Static Analysis (1/2)

Suppose that

- We are interested in the value ranges of variables.
- How to finitely estimate $[\![P]\!]$ for the property?

You may, intuitively:

```
     x = readInt;
1:
     while (x ≤ 99)
2:
          x++;
3:
     end
4:
```

Capture the dynamics by abstract equations; solve; reason.

$$
\begin{array}{rcl}
x_1 & = & [-\infty, +\infty] \; or \; x_3 \\
x_2 & = & x_1 \; and \; [-\infty, 99] \\
x_3 & = & x_2 \dot{+} 1 \\
x_4 & = & x_1 \; and \; [100, +\infty]
\end{array}
$$

Need Formal Frameworks of Static Analysis (2/2)

Abstract Interpretation [CousotCousot]: a powerful design theory

- How to derive correct yet arbitrarily precise equations?
  - Non-obvious: ptrs, heap, exns, high-order ftns, etc.

```
x = readInt;
while (x ≤ 99)
    x++;
end
```

$\overset{\text{how?}}{\Longrightarrow}$

$$\begin{array}{lcl} x_1 & = & [-\infty,+\infty] \ or \ x_3 \\ x_2 & = & x_1 \ and \ [-\infty,99] \\ x_3 & = & x_2 \dot{+} 1 \\ x_4 & = & x_1 \ and \ [100,+\infty] \end{array}$$

  - Define an abstract semantics function $\hat{F}$ s.t. $\cdots$
- How to solve the equations in a finite time?

$$\begin{array}{lcl} x_1 & = & [-\infty,+\infty] \ or \ x_3 \\ x_2 & = & x_1 \ and \ [-\infty,99] \\ x_3 & = & x_2 \dot{+} 1 \\ x_4 & = & x_1 \ and \ [100,\infty] \end{array} \quad \overset{\text{how?}}{\Longrightarrow} \quad \begin{array}{lcl} x_1 & = & [-\infty,+\infty] \\ x_2 & = & [-\infty,99] \\ x_3 & = & [-\infty,100] \\ x_4 & = & [100,+\infty] \end{array}$$

  - Fixpoint iterations for an upperbound of $\mathit{fix}\hat{F}$

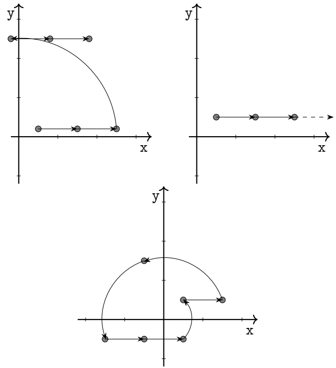
Outline

Example Language

$$
\begin{array}{lcll}
\mathtt{p} & ::= & \mathtt{init}(\mathfrak{R}) & \text{initialization, with a state in } \mathfrak{R} \\
& | & \mathtt{translation}(u, v) & \text{translation by vector } (u, v) \\
& | & \mathtt{rotation}(u, v, \theta) & \text{rotation by center } (u, v) \text{ and angle } \theta \\
& | & \mathtt{p}\ ;\ \mathtt{p} & \text{sequence of operations} \\
& | & \{\mathtt{p}\}\mathtt{or}\{\mathtt{p}\} & \text{non-deterministic choice} \\
& | & \mathtt{iter}\{\mathtt{p}\} & \text{non-deterministic iterations}
\end{array}
$$

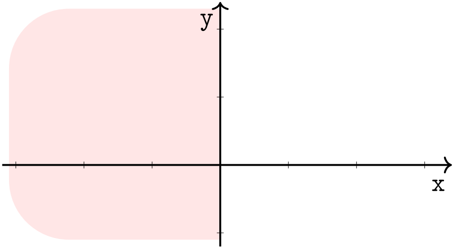
Example (Semantics)

$\mathtt{init}([0,1]\times[0,1]);$
$\mathtt{translation}(1,0);$
$\mathtt{iter}\{$
$\quad\{$
$\qquad\mathtt{translation}(1,0)$
$\quad\}\mathtt{or}\{$
$\qquad\mathtt{rotation}(0,0,90^\circ)$
$\quad\}$
$\}$

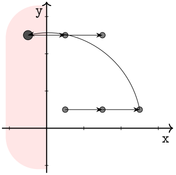


Analysis Goal Is Safety Property: Reachability

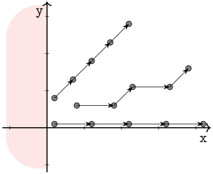
Analyze the set of reachable points, to check if the set intersects with a no-fly zone. Suppose that the no-fly zone is:



Correct or Incorrect Executions



(a) An incorrect execution

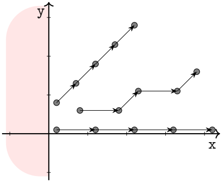


(b) Correct executions

An Example Safe Program

Example

$$
\begin{array}{l}
\texttt{init}([0,1]\times[0,1]);\\
\texttt{iter}\{\\
\quad\{\\
\qquad \texttt{translation}(1,0);\\
\quad\}\texttt{or}\{\\
\qquad \texttt{translation}(0.5,0.5);\\
\quad\}\\
\}
\end{array}
$$



How to Finitely Over-Approximate the Set of Reachable Points?

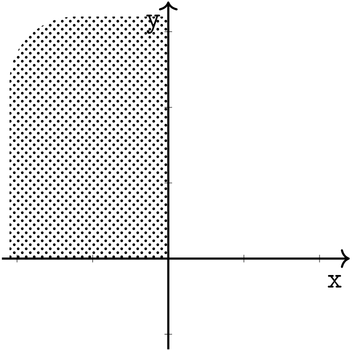
Definition (Abstraction)

We call *abstraction* a set $\mathcal{A}$ of logical properties of program states, which are called *abstract properties* or *abstract elements*. A set of abstract properties is called an *abstract domain*.

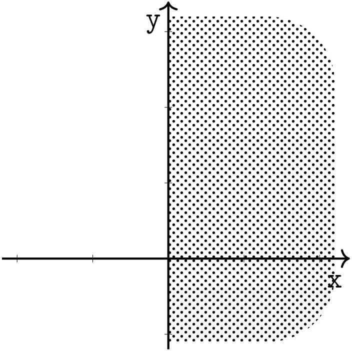
Definition (Concretization)

Given an abstract element a of $\mathcal{A}$, we call *concretization* the set of program states that satisfy it. We denote it by $\gamma(a)$.

Abstraction Example 1: Signs Abstraction



(c) Concretization of $[\mathtt{x} \leq 0, \mathtt{y} \geq 0]$

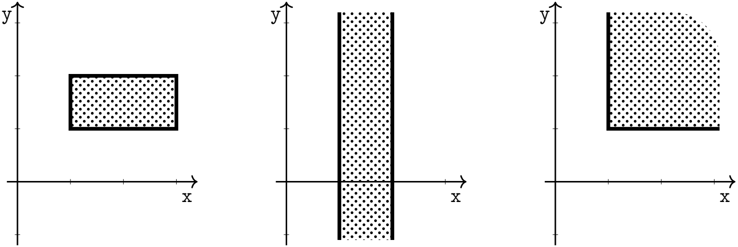


(d) Concretization of $[\mathtt{x} \geq 0]$

Figure: Signs abstraction

Abstraction Example 2: Interval Abstraction

The abstract elements: conjunctions of non-relational inequality constraints: $c_1 \leq \mathrm{x} \leq c_2$, $c_1' \leq \mathrm{y} \leq c_2'$



(a) Concretization of $[1 \leq \mathrm{x} \leq 3, 1 \leq \mathrm{y} \leq 2]$

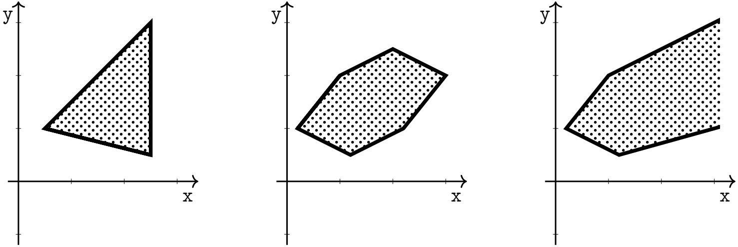
(b) Concretization of $[1 \leq \mathrm{x} \leq 2]$

(c) Concretization of $[1 \leq \mathrm{x}, 1 \leq \mathrm{y}]$

Figure: Intervals abstraction

Abstraction Example 3: Convex Polyhedra Abstraction

The abstract elements: conjunctions of linear inequality constraints:
$c_1\mathrm{x} + c_2\mathrm{y} \leq c_3$



(a) Concretization of a_0 (b) Concretization of a_1 (c) Concretization of a_2

Figure: Convex polyhedra abstraction

An Example Program, Again

Example

```
init([0,1] × [0,1]);
iter{
    {
        translation(1,0);
    }or{
        translation(0.5,0.5);
    }
}
```

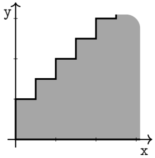
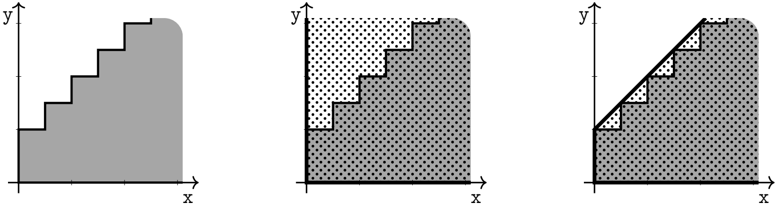


Figure: Reachable states

Abstractions of the Semantics of the Example Program



(a) Reachable states (b) Intervals abstraction (c) Convex polyhedra abstraction

Figure: Program's reachable states and abstraction

Sound Analysis Function for the Example Language

- Input: a program p and an abstract area a (pre-state)
- Output: an abstract area a' (post-state)

Definition (Sound analysis)

An `analysis` is sound if and only if **it captures the real executions of the input program**.

If an execution of p moves a point $(\mathtt{x}, \mathtt{y})$ to point $(\mathtt{x}', \mathtt{y}')$,
then for all abstract element a such that $(\mathtt{x}, \mathtt{y}) \in \gamma(a)$,

$$(\mathtt{x}', \mathtt{y}') \in \gamma(\mathtt{analysis}(\mathtt{p}, a))$$

Sound Analysis Function as a Diagram

If
$$\begin{tikzcd} a_{\mathrm{pre}} & \\ (\mathtt{x}, \mathtt{y}) \arrow[u, "\text{abstraction}"] \arrow[r, "\text{run } \mathtt{p}"] & (\mathtt{x}', \mathtt{y}') \end{tikzcd}$$
then
$$\begin{tikzcd} a_{\mathrm{pre}} \arrow[r, "\text{analyze } \mathtt{p}"] & a_{\mathrm{post}} = \mathtt{analysis}(\mathtt{p}, a_{\mathrm{pre}}) \\ (\mathtt{x}, \mathtt{y}) \arrow[u, "\text{abstraction}"] \arrow[r, "\text{run } \mathtt{p}"] & (\mathtt{x}', \mathtt{y}') \arrow[u, "\text{abstraction}"'] \end{tikzcd}$$

Figure: Sound analysis of a program p

Abstract Semantics Computation

Recall the example language

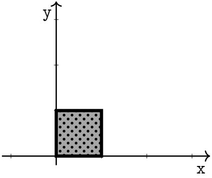
$$
\begin{array}{lcll}
\mathtt{p} & ::= & \mathtt{init}(\mathfrak{R}) & \text{initialization, with a state in } \mathfrak{R} \\
& | & \mathtt{translation}(u, v) & \text{translation by vector } (u, v) \\
& | & \mathtt{rotation}(u, v, \theta) & \text{rotation defined by center } (u, v) \text{ and angle } \theta \\
& | & \mathtt{p}\ ;\ \mathtt{p} & \text{sequence of operations} \\
& | & \{\mathtt{p}\}\mathtt{or}\{\mathtt{p}\} & \text{non-deterministic choice} \\
& | & \mathtt{iter}\{\mathtt{p}\} & \text{non-deterministic iterations}
\end{array}
$$

Approach

A sound analysis for a program is constructed by computing sound abstract semantics of the program's components.

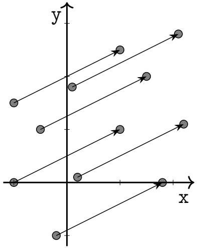
Abstract Semantics Computation: $\mathtt{init}(\mathfrak{R})$

- Select, if any, the best abstraction of the region $\mathfrak{R}$.
- For the example program with the intervals or convex polyhedra abstract domains, analysis of $\mathtt{init}([0,1] \times [0,1])$ is

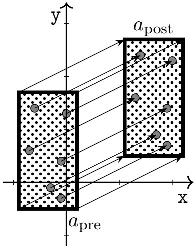


$$\mathtt{analysis}(\mathtt{init}(\mathfrak{R}), a) = \text{best abstraction of the region } \mathfrak{R}$$

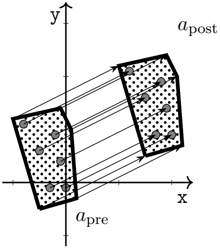
Abstract Semantics Computation: $\texttt{translation}(u, v)$



(a) Concrete semantics



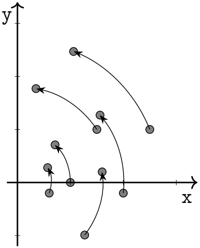
(b) Intervals



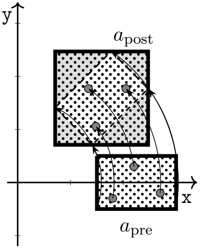
(c) Convex polyhedra

$$\texttt{analysis}(\texttt{translation}(u, v), a) = \left\{ \begin{array}{l} \text{return an abstract state that contains} \\ \text{the translation of } a \end{array} \right.$$

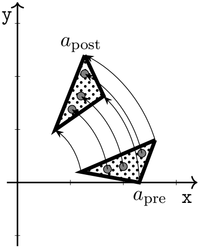
Abstract Semantics Computation: $\texttt{rotation}(u, v, \theta)$



(d) Concrete semantics



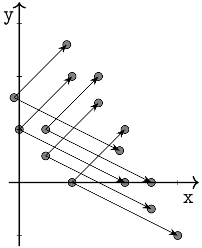
(e) Intervals



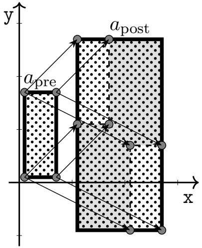
(f) Convex polyhedra

$$\texttt{analysis}(\texttt{rotation}(u, v, \theta), a) = \begin{cases} \text{return an abstract state that contains} \\ \text{the rotation of } a \end{cases}$$

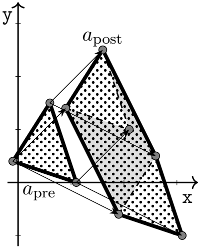
Abstract Semantics Computation: $\{\mathtt{p}\}\mathtt{or}\{\mathtt{p}\}$



(g) Concrete semantics



(h) Intervals



(i) Convex polyhedra

$$\texttt{analysis}(\{\mathtt{p}_0\}\mathtt{or}\{\mathtt{p}_1\}, a) = \mathtt{union}(\texttt{analysis}(\mathtt{p}_1, a), \texttt{analysis}(\mathtt{p}_0, a))$$

Abstract Semantics Computation: $\texttt{p}_0$; $\texttt{p}_1$

$$\texttt{analysis}(\texttt{p}_0;\texttt{p}_1, a) = \texttt{analysis}(\texttt{p}_1, \texttt{analysis}(\texttt{p}_0, a))$$

Abstract Semantics Computation: $\texttt{iter}\{\texttt{p}\}$ (1/5)

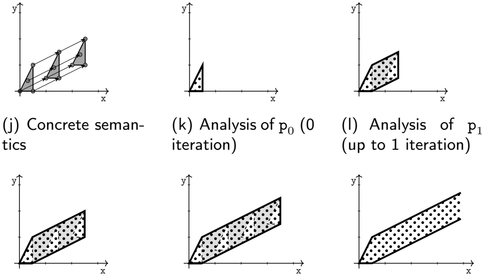
$\texttt{iter}\{\texttt{p}\}$ is equivalent to

$$
\begin{array}{l}
\{\} \\
\texttt{or}\{\texttt{p}\} \\
\texttt{or}\{\texttt{p}; \texttt{p}\} \\
\texttt{or}\{\texttt{p}; \texttt{p}; \texttt{p}\} \\
\texttt{or}\{\texttt{p}; \texttt{p}; \texttt{p}; \texttt{p}\} \\
\vdots
\end{array}
$$

Abstract Semantics Computation: iter{p} (2/5)

Example (Abstract iteration)

```
init({(x, y) | 0 ≤ y ≤ 2x and x ≤ 0.5});
iter{
    translation(1, 0.5)
}
```



(j) Concrete semantics

(k) Analysis of p_0 (0 iteration)

(l) Analysis of p_1 (up to 1 iteration)

(m) Analysis of p_2 (up to 2 iterations)

(n) Analysis of p_3 (up to 3 iterations)

(o) Expected result

Figure: Abstract iteration

Abstract Semantics Computation: $\mathtt{iter}\{\mathtt{p}\}$ (3/5)

Recall

$$\begin{array}{rcl} \mathtt{iter}\{\mathtt{p}\} & = & \{\}\ \mathtt{or}\ \{\mathtt{p}\}\ \mathtt{or}\ \{\mathtt{p};\mathtt{p}\}\ \mathtt{or} \cdots \\ & = & \lim_i \mathtt{p}_i \end{array}$$

where

$$\mathtt{p}_0 = \{\} \qquad \mathtt{p}_{k+1} = \mathtt{p}_k\ \mathtt{or}\ \{\mathtt{p}_k;\mathtt{p}\}$$

Hence,

$$\mathtt{analysis}(\mathtt{iter}\{\mathtt{p}\}, a) \quad = \quad \left\{ \begin{array}{l} \mathtt{R} \leftarrow a; \\ \text{repeat} \\ \qquad \mathtt{T} \leftarrow \mathtt{R}; \\ \qquad \mathtt{R} \leftarrow \mathtt{widen}(\mathtt{R}, \mathtt{analysis}(\mathtt{p}, \mathtt{R})); \\ \text{until } \mathtt{inclusion}(\mathtt{R}, \mathtt{T}) \\ \text{return } \mathtt{T}; \end{array} \right.$$

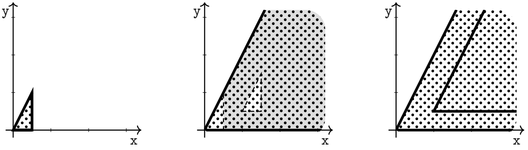
$$\text{operator } \mathtt{widen} \quad \left\{ \begin{array}{l} \text{over approximates unions} \\ \text{enforces finite convergence} \end{array} \right.$$

Abstract Semantics Computation: iter{p} (4/5)

Example (Abstract iteration with widening)

$$
\begin{array}{l}
\texttt{init}(\{(\mathrm{x},\mathrm{y}) \mid 0 \le \mathrm{y} \le 2\mathrm{x} \text{ and } \mathrm{x} \le 0.5\});\\
\texttt{iter}\{\\
\quad \texttt{translation}(1, 0.5)\\
\}
\end{array}
$$

- The constraints $0 \le \mathrm{y}$ and $\mathrm{y} \le 2\mathrm{x}$ are stable after iteration 1; thus, they are preserved.
- The constraint $\mathrm{x} \le 0.5$ is not preserved; thus, it is discarded.



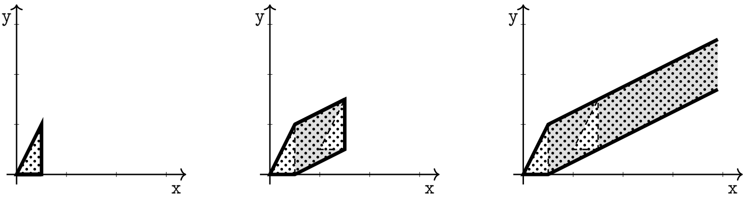
(a) Iteration 0 (b) Iteration 1 (c) Iteration 2 and limit

Figure: Abstract iteration with widening

Abstract Semantics Computation: $\texttt{iter}\{\texttt{p}\}$ (5/5)

Example (Loop unrolling)

$$\texttt{init}(\{(\mathrm{x},\mathrm{y}) \mid 0 \le \mathrm{y} \le 2\mathrm{x} \text{ and } \mathrm{x} \le 0.5\});$$
$$\{\}\ \texttt{or}\ \{\ \texttt{translation}(1, 0.5)\ \};$$
$$\texttt{iter}\{\ \texttt{translation}(1, 0.5)\ \}$$



(a) Iteration 0 (b) Iteration 1, `union` (c) Iteration 2, `widen`, limit

Figure: Abstract iteration with widening and unrolling

Abstract Semantics Function analysis at a Glance

The $\texttt{analysis}(\mathrm{p}, a)$ is finitely computable and sound.

$$
\begin{array}{rcl}
\texttt{analysis}(\texttt{init}(\mathfrak{R}), a) & = & \text{best abstraction of the region } \mathfrak{R} \\
\texttt{analysis}(\texttt{translation}(u, v), a) & = & \left\{ \begin{array}{l} \text{return an abstract state that contains} \\ \text{the translation of } a \end{array} \right. \\
\texttt{analysis}(\texttt{rotation}(u, v, \theta), a) & = & \left\{ \begin{array}{l} \text{return an abstract state that contains} \\ \text{the rotation of } a \end{array} \right. \\
\texttt{analysis}(\{\mathrm{p}_0\}\texttt{or}\{\mathrm{p}_1\}, a) & = & \texttt{union}(\texttt{analysis}(\mathrm{p}_1, a), \texttt{analysis}(\mathrm{p}_0, a)) \\
\texttt{analysis}(\mathrm{p}_0; \mathrm{p}_1, a) & = & \texttt{analysis}(\mathrm{p}_1, \texttt{analysis}(\mathrm{p}_0, a)) \\
\texttt{analysis}(\texttt{iter}\{\mathrm{p}\}, a) & = & \left\{ \begin{array}{l} \mathrm{R} \leftarrow a; \\ \text{repeat} \\ \qquad \mathrm{T} \leftarrow \mathrm{R}; \\ \qquad \mathrm{R} \leftarrow \texttt{widen}(\mathrm{R}, \texttt{analysis}(\mathrm{p}, \mathrm{R})); \\ \text{until } \texttt{inclusion}(\mathrm{R}, \mathrm{T}) \\ \text{return } \mathrm{T}; \end{array} \right.
\end{array}
$$

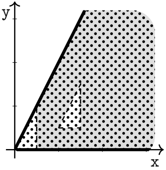
Sound analysis

If an execution of p from a state (x, y) generates the state $(\mathrm{x}', \mathrm{y}')$, then for all abstract element a such that $(\mathrm{x}, \mathrm{y}) \in \gamma(a)$,

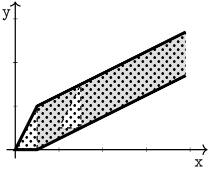
$$(\mathrm{x}', \mathrm{y}') \in \gamma(\texttt{analysis}(\mathrm{p}, a))$$

Verification of the Property of Interest

- Does program compute a point inside no-fly zone $\mathfrak{D}$?
- Need to collect the set of reachable points.
- Run `analysis`$(\mathrm{p}, -)$ and collect all points $\mathfrak{R}$ from every call to `analysis`.
- Since `analysis` is sound, the result is an over approx. of the reachable points.
- **If $\mathfrak{R} \cap \mathfrak{D} = \emptyset$, then p is verified. Otherwise, we don't know.**



(a) A $\mathfrak{R}$



(b) A more precise $\mathfrak{R}$

Semantics Style: Compositional Versus Transitional

- Compositional semantics function `analysis`:
  - Semantics of `p` is defined by the semantics of the sub-parts of `p`.

$$[\![AB]\!] = \cdots [\![A]\!] \cdots [\![B]\!] \cdots$$

  - Proving its soundness is thus by structural induction on `p`.
- For some realistic programming languages, even defining their compositional ("denotational") semantics is a hurdle.
  - gotos, exceptions, function calls

Transitional-style ("operational") semantics avoids the hurdle

$$[\![AB]\!] = \{s_0 \hookrightarrow s_1 \hookrightarrow \cdots, \cdots\}$$

Example Language, Again

p	::=	init($\mathfrak{R}$)	initialization, with a state in $\mathfrak{R}$
	$\mid$	translation(u, v)	translation by vector (u, v)
	$\mid$	rotation(u, v, θ)	rotation by center (u, v) and angle θ
	$\mid$	p ; p	sequence of operations
	$\mid$	{p}or{p}	non-deterministic choice
	$\mid$	iter{p}	non-deterministic iterations

Semantics as State Transitions

Definition (Transitional semantics)

An execution of a program is a sequence of transitions between states.

- a state is a pair (l, p) of statement label l and an (x,y) point p.
- a single transition

$$(l, p) \hookrightarrow (l', p')$$

whenever the program statement at l moves the point p to p'.

$$s_1 \hookrightarrow s_2 \hookrightarrow s_5 \hookrightarrow s_3 \hookrightarrow s_8 \hookrightarrow \cdots$$
$$s_6 \hookrightarrow s_7 \hookrightarrow s_8 \hookrightarrow s_3 \hookrightarrow s_4$$
$$s_9 \hookrightarrow s_{10} \hookrightarrow s_8 \hookrightarrow s_{11} \hookrightarrow s_8 \hookrightarrow s_{11} \hookrightarrow s_{13}$$
$$s_{12} \hookrightarrow s_7 \hookrightarrow s_2 \hookrightarrow s_3 \hookrightarrow s_4 \hookrightarrow s_{14}$$

States $s_1, s_6, s_9,$ and s_{12} are initial states.

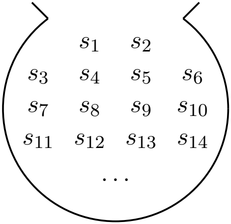
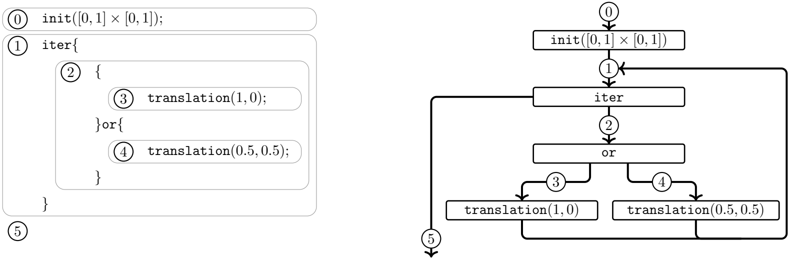


Figure: Transition sequences and the set of occurring states

Statement Labels

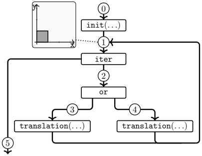


(a) Text view, with labels

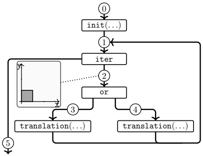
(b) Graph view, with labels

Figure: Example program with statement labels

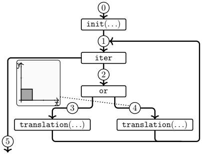
States in a Transition Sequence



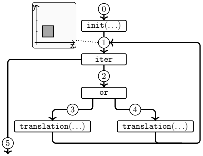
(a) State $(1, p_1)$



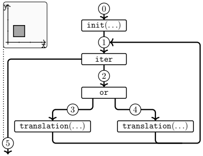
(b) State $(2, p_1)$



(c) State $(4, p_1)$



(d) State $(1, p_3)$

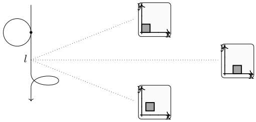


(e) State $(5, p_3)$

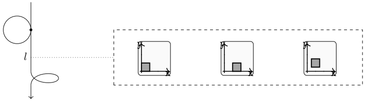
Reachability Problem and Abstraction of States

- Reachability problem: compute the set of all states that can occur during all transition sequences of the input program.
- An abstract state is a set of pairs of statement labels and abstract pre conditions.

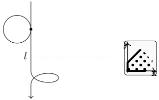
Collection of all states



Statement-wise collection:



Statement-wise abstraction:



Abstract State Transition

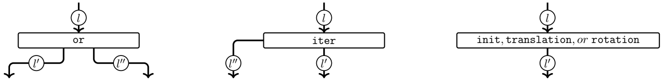
$$\begin{array}{rl} \textit{Step}^\sharp: & \text{a set of pairs of labels and abstract pre conditions} \\ & \mapsto \\ & \text{a set of pairs of labels and abstract post conditions} \end{array}$$

is

$$\textit{Step}^\sharp(X) = \{x' \mid x \in X, x \hookrightarrow^\sharp x'\}$$

where

$$\begin{array}{rcl} (\texttt{or}_l, a_{\text{pre}}) & \hookrightarrow^\sharp & (\texttt{next}(l), a_{\text{pre}}) \\ (\texttt{iter}_l, a_{\text{pre}}) & \hookrightarrow^\sharp & (\texttt{next}(l), a_{\text{pre}}) \\ (\texttt{p}_l, a_{\text{pre}}) & \hookrightarrow^\sharp & (\texttt{next}(l), \texttt{analysis}(\texttt{p}_l, a_{\text{pre}})) \end{array}$$



Analysis by Global Iterations

The analysis goal is to accumulate from the initial abstract state I:

$$\mathit{Step}^{\sharp^0}(I) \cup \mathit{Step}^{\sharp^1}(I) \cup \mathit{Step}^{\sharp^2}(I) \cup \cdots$$

which is the limit C_∞ of $C_i = \mathit{Step}^{\sharp^0}(I) \cup \mathit{Step}^{\sharp^1}(I) \cup \cdots \cup \mathit{Step}^{\sharp^i}(I)$ where

$$C_{k+1} = C_k \cup \mathit{Step}^\sharp(C_k).$$

Thus the analysis algorithm should iterate the operation

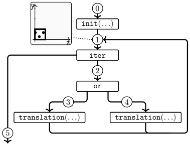
$$\mathtt{C} \leftarrow \mathtt{C} \cup \mathit{Step}^\sharp(\mathtt{C})$$

from I until stable:

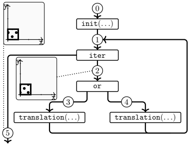
$$\mathtt{analysis}_T(\mathtt{p}, I) = \begin{cases} \mathtt{C} \leftarrow I \\ \text{repeat} \\ \qquad \mathtt{R} \leftarrow \mathtt{C} \\ \qquad \mathtt{C} \leftarrow \mathtt{widen}_T(\mathtt{C}, \mathit{Step}^\sharp(\mathtt{C})) \\ \text{until } \mathtt{inclusion}_T(\mathtt{C}, \mathtt{R}) \\ \text{return } \mathtt{R} \end{cases}$$

where $\mathtt{widen}_T$ over-approximates unions and enforces finite convergence.

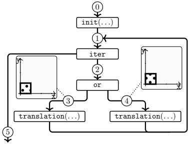
Analysis in Action



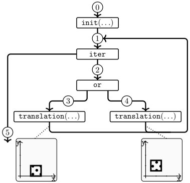
(f) State $(1, a_1)$



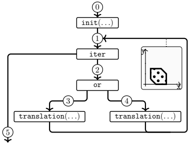
(g) States $(2, a_1) and (5, a_1)$



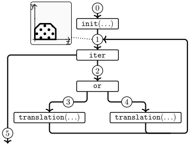
(h) States $(3, a_1) and (4, a_1)$



(i) States $(1, a_2) and (1, a_3)$



(j) State $(1, \mathtt{union}(\{a_2, a_3\}))$



(k) State $(1, \mathtt{union}(\{a_1, a_2, a_3\}))$

Principles of a Static Analysis, Sketchy

- Selection of the semantics and properties of interest:
  - define the behaviors of programs
  - define the properties that need to be verified
  - formal definitions
- Choice of the abstraction:
  - define the space of abstract elements over which the abstract semantics is defined
  - define what the abstract elements mean
  - define abstract semantics and prove its soundness
- Derivation of the analysis algorithms from the semantics and from the abstraction:
  - algorithm follows the semantic formalism in use
  - e.g., compositional algorithm in the style of program interpreter
  - e.g., transitional algorithm by a monolithic, global iterations

Outline

Transitional Semantics

State transition sequence

$$s_0 \hookrightarrow s_1 \hookrightarrow s_2 \hookrightarrow \cdots$$

where $\hookrightarrow$ is a transition relation between states $\mathbb{S}$

$$\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$$

A state $s \in \mathbb{S}$ of the program is a pair (l, m) of a program label l and the machine state m at that program label during execution.

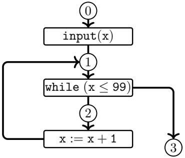
Concrete Transition Sequence

Example

Consider the following program

$$\begin{array}{l}\texttt{input}(\texttt{x});\\ \texttt{while}\ (\texttt{x} \leq 99)\\ \qquad \{\texttt{x} := \texttt{x} + 1\}\end{array}$$

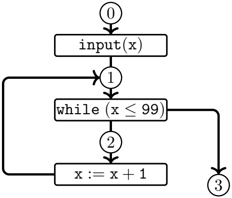
Let labels be "program points". Such labeled representations of this program in graph is



Let the initial state be the empty memory $\emptyset$. Some transition sequences are:

For input 100: $(0, \emptyset) \hookrightarrow (1, x \mapsto 100) \hookrightarrow (3, x \mapsto 100)$.
For input 99: $(0, \emptyset) \hookrightarrow (1, x \mapsto 99) \hookrightarrow (2, x \mapsto 99) \hookrightarrow (1, x \mapsto 100) \hookrightarrow (3, x \mapsto 100)$.
For input 0: $(0, \emptyset) \hookrightarrow (1, x \mapsto 0) \hookrightarrow (2, x \mapsto 0) \hookrightarrow (1, x \mapsto 1) \hookrightarrow \cdots \hookrightarrow (3, x \mapsto 100)$.

Reachable States



Assume that the possible inputs are 0, 99, and 100. Then, the set of all reachable states are the set of states occurring in the three transition sequences:

$$
\begin{array}{ll}
& \{(0,\emptyset),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
\cup & \{(0,\emptyset),(1,x\mapsto 99),(2,x\mapsto 99),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
\cup & \{(0,\emptyset),(1,x\mapsto 0),(2,x\mapsto 0),(1,x\mapsto 1),\cdots,(2,x\mapsto 99),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
= & \{(0,\emptyset),(1,x\mapsto 0),\cdots,(1,x\mapsto 100),(2,x\mapsto 0),\cdots,(2,x\mapsto 99),(3,x\mapsto 100)\}
\end{array}
$$

Concrete Semantics: the Set of Reachable States (1/3)

Given a program, let I be the set of its initial states and *Step* be the powerset-lifted version of $\hookrightarrow$:

$$\begin{array}{l} \mathit{Step} : \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ \mathit{Step}(X) = \{s' \mid s \hookrightarrow s', s \in X\} \end{array}$$

The set of reachable states is

$$I \;\cup\; \mathit{Step}^1(I) \;\cup\; \mathit{Step}^2(I) \;\cup\; \cdots .$$

which is, equivalently, the limit of C_is

$$\begin{array}{rcl} C_0 & = & I \\ C_{i+1} & = & I \;\cup\; \mathit{Step}(C_i) \end{array}$$

which is, the least solution of

$$X = I \;\cup\; \mathit{Step}(X).$$

Concrete Semantics: the Set of Reachable States (2/3)

The least solution of

$$X = I \cup \mathit{Step}(X)$$

is also called *the least fixpoint* of F

$$F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$$
$$F(X) = I \cup \mathit{Step}(X)$$

written as

$$\mathbf{lfp} F.$$

Theorem (Least fixpoint)

The least fixpoint $\mathbf{lfp} F$ *of* $F(X) = I \cup \mathit{Step}(X)$ *is*

$$\bigcup_{i \geq 0} F^i(\emptyset)$$

where $F^0(X) = X$ *and* $F^{n+1}(X) = F(F^n(X))$.

Concrete Semantics: the Set of Reachable States (3/3)

Definition (Concrete semantics, the set of reachable states)

Given a program, let $\mathbb{S}$ be the set of states and $\hookrightarrow$ be the one-step transition relation $\subseteq \mathbb{S} \times \mathbb{S}$. Let I be the set of its initial states and *Step* be the powerset-lifted version of $\hookrightarrow$:

$$\begin{array}{l} \mathit{Step} : \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ \mathit{Step}(X) = \{s' \mid s \hookrightarrow s', s \in X\}. \end{array}$$

Then the concrete semantics of the program, the set of all reachable states from I, is defined as the least fixpoint $\mathbf{lfp}F$ of F

$$F(X) = I \cup \mathit{Step}(X).$$

Analysis Goal

Program-label-wise reachability

For each program label we want to know the set of memories that can occur at that label during executions of the input program.

- labels: "partitioning indices"
- e.g., statement labels as in programs, statement labels after loop unrolling, statement labels after function inlining

Abstract Semantics

Define the abstract semantics "homomorphically":

$$F : \wp(\mathbb{S}) \to \wp(\mathbb{S}) \qquad\qquad F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp$$
$$F(X) = I \cup \mathit{Step}(X) \qquad\qquad F^\sharp(X^\sharp) = I^\sharp \cup^\sharp \mathit{Step}^\sharp(X^\sharp)$$

The forthcoming framework will guide us

- conditions for $\mathbb{S}^\sharp$ and $F^\sharp$
- so that the abstract semantics is finitely computable and is an upper-approximation of concrete semantics **lfp**F.

Abstraction of the Semantic Domain $\wp(\mathbb{S})$ (1/2)

$$\wp(\mathbb{S}) \quad \text{where} \quad \mathbb{S} = \mathbb{L} \times \mathbb{M}$$

Label-wise (two-step) abstraction of states:

$$\begin{array}{ccccc} \text{set of states} & \text{to} & \text{label-wise collect} & \text{to} & \text{label-wise abstraction} \\ \wp(\mathbb{L} \times \mathbb{M}) & \xrightarrow{\text{abstraction}} & \mathbb{L} \to \wp(\mathbb{M}) & \xrightarrow{\text{abstraction}} & \mathbb{L} \to \mathbb{M}^{\sharp}. \end{array}$$

Abstraction of the Semantic Domain $\wp(\mathbb{S})$ (2/2)

$$\wp(\mathbb{L}\times\mathbb{M}) \ni \quad \text{collection of all states} \quad \begin{cases} (0,m_0),(0,m_0'),\cdots, & \text{at } 0 \\ (1,m_1),(1,m_1'),\cdots, & \text{at } 1 \\ \quad\vdots & \\ (n,m_n),(n,m_n'),\cdots. & \text{at } n \end{cases}$$

$$\mathbb{L}\to\wp(\mathbb{M}) \ni \quad \text{label-wise collection} \quad \begin{cases} (0,\{m_0,m_0',\cdots\}) \\ (1,\{m_1,m_1',\cdots\}) \\ \quad\vdots \\ (n,\{m_n,m_n',\cdots\}) \end{cases}$$

$$\mathbb{L}\to\mathbb{M}^\sharp \ni \quad \text{label-wise abstraction} \quad \begin{cases} (0,M_0^\sharp) \\ (1,M_1^\sharp) \\ \quad\vdots \\ (n,M_n^\sharp) \end{cases}$$

Each $M_l^\sharp$ over-approximates the set $\{m_l,m_l',\cdots\}$ collected at label l.

Preliminary for Abstract Domains (1/3)

- Define an abstract domain as a *CPO*
  - a partial order set
  - has a least element $\perp$
  - has a least-upper bound for every *chain*
- An abstract domain as $\sqcup$-semilattices also work.

Preliminary for Abstract Domains (2/3)

Abstract and concrete domains are structured "consistently".

Definition (Galois connection)

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

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

We write such a pair as follows:

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

Preliminary for Abstract Doamins (3/3)

Galois-connection properties we rely on:
For

$$(\mathbb{C}, \subseteq) \xleftarrow[\alpha]{\gamma} \!\!\!\!\!\!\!\!\!\!\!\xrightarrow{} (\mathbb{A}, \sqsubseteq)$$

- α and γ are monotone functions
- $\forall c \in \mathbb{C},\ c \subseteq \gamma(\alpha(c))$
- $\forall a \in \mathbb{A},\ \alpha(\gamma(a)) \sqsubseteq a$
- If both $\mathbb{C}$ and $\mathbb{A}$ are CPOs, then α is continuous.

(Proofs are in the supplementary note.)

Abstract Domains (1/2)

Design an abstract domain as a CPO that is Galois-connected with the concrete domain:

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \xleftrightarrow[\alpha]{\gamma} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq).$$

- Abstraction α defines how each concrete elmt (set of concrete states) is abstracted into an abstract elmt.
- Concretization γ defines the set of concrete states implied by each abstract state.
- Partial order $\sqsubseteq$ is the label-wise order:

$$a^\sharp \sqsubseteq b^\sharp \quad \text{iff} \quad \forall l \in \mathbb{L} : a^\sharp(l) \sqsubseteq_M b^\sharp(l)$$

where $\sqsubseteq_M$ is the partial order of $\mathbb{M}^\sharp$.

Abstract Domains (2/2)

The above Galois connection (abstraction)

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \xleftarrow[\alpha]{\gamma} \xrightarrow{} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq).$$

composes two Galois connections:

$$\begin{array}{l} (\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \\ \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} \quad (\mathbb{L} \to \wp(\mathbb{M}), \sqsubseteq) \quad (\sqsubseteq \text{ is the label-wise } \subseteq) \\ \qquad \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq) \quad (\sqsubseteq \text{ is the label-wise } \sqsubseteq_M) \end{array}$$

$$\alpha_0 \left\{ \begin{array}{l} (0, m_0), (0, m_0'), \cdots, \\ \quad \vdots \\ (n, m_n), (n, m_n'), \cdots \end{array} \right\} = \left\{ \begin{array}{l} (0, \{m_0, m_0', \cdots\}), \\ \quad \vdots \\ (n, \{m_n, m_n', \cdots\}) \end{array} \right\}$$

$$\alpha_1 \left\{ \begin{array}{l} (0, \{m_0, m_0', \cdots\}), \\ \quad \vdots \\ (n, \{m_n, m_n', \cdots\}) \end{array} \right\} = \left\{ \begin{array}{l} (0, M_0^\sharp), \\ \quad \vdots \\ (n, M_n^\sharp) \end{array} \right\}$$

Thus, boils down to

$$(\wp(\mathbb{M}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}^\sharp, \sqsubseteq_M).$$

Abstract Semantic Functions

Let

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \xleftrightarrows[\alpha]{\gamma} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq).$$

A concrete semantic function F

$$\begin{aligned}
\mathbb{S} &= \mathbb{L} \times \mathbb{M} \\
F &: \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\
F(X) &= I \cup \textit{Step}(X) \\
\textit{Step} &= \breve{\wp}(\hookrightarrow) \\
\hookrightarrow &\subseteq (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})
\end{aligned}$$

An abstract semantic function $F^\sharp$

$$\begin{aligned}
\mathbb{S}^\sharp &= \mathbb{L} \to \mathbb{M}^\sharp \\
F^\sharp &: \mathbb{S}^\sharp \to \mathbb{S}^\sharp \\
F^\sharp(X^\sharp) &= \alpha(I) \cup^\sharp \textit{Step}^\sharp(X^\sharp) \\
\textit{Step}^\sharp &= \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp) \\
\hookrightarrow^\sharp &\subseteq (\mathbb{L} \times \mathbb{M}^\sharp) \times (\mathbb{L} \times \mathbb{M}^\sharp)
\end{aligned}$$

with relations $\hookrightarrow$ and $\hookrightarrow^\sharp$ being functions

As of $\mathit{Step}^\sharp = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp)$

$\mathit{Step}^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$

- Abstract transition $\breve{\wp}(\hookrightarrow^\sharp)$:
  - a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp \quad \mapsto \quad$ a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp$
- Paritioning π:
  - a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp \quad \mapsto \quad$ a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp)$
- Joining $\wp(\mathrm{id}, \sqcup_M)$:
  - a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp) \quad \mapsto \quad$ an abstract state $\in \mathbb{L} \to \mathbb{M}^\sharp$

Example

Suppose the program has two labels l_1 and l_2. That is, $\mathbb{L} = \{l_1, l_2\}$. Given an abstract state $\{(l_1, M_1^\sharp), (l_2, M_2^\sharp)\}$, *Step*$^\sharp$ first applies $\breve{\wp}(\hookrightarrow^\sharp)$ to it:

$$\hookrightarrow^\sharp(l_1, M_1^\sharp) \cup \hookrightarrow^\sharp(l_2, M_2^\sharp).$$

Suppose $\hookrightarrow^\sharp(l_1, M_1^\sharp)$ returns $\{(l_1, {M'}_1^\sharp), (l_2, {M''}_1^\sharp)\}$ and $\hookrightarrow^\sharp(l_2, M_2^\sharp)$ returns $\{(l_1, {M'}_2^\sharp)\}$. Then the result is

$$\{(l_1, {M'}_1^\sharp), (l_2, {M''}_1^\sharp), (l_1, {M'}_2^\sharp)\}.$$

The subsequent application of the operator π partitions the result by labels into

$$\{(l_1, \{{M'}_1^\sharp, {M'}_2^\sharp\}), (l_2, \{{M''}_1^\sharp\})\}.$$

The final organization operation $\wp(\mathrm{id}, \sqcup_M)$ returns the post abstract state $\in \mathbb{L} \to \mathbb{M}^\sharp$:

$$\{(l_1, {M'}_1^\sharp \sqcup_M {M'}_2^\sharp), (l_2, {M''}_1^\sharp)\}.$$

Conditions for Sound $\hookrightarrow^\sharp$ and $\cup^\sharp$

- sound condition for $\hookrightarrow^\sharp$:

$$\breve{\wp}(\hookrightarrow) \circ \gamma \ \subseteq \ \gamma \circ \breve{\wp}(\hookrightarrow^\sharp)$$

- sound condition for $\cup^\sharp$:

$$\cup \circ (\gamma, \gamma) \ \subseteq \ \gamma \circ \cup^\sharp$$

$$\begin{tikzcd} X^\sharp \arrow[r, "\breve{\wp}(\hookrightarrow^\sharp)"] \arrow[d, "\gamma"'] & Y^\sharp \arrow[d, "\gamma"] \\ X \arrow[r, "\breve{\wp}(\hookrightarrow)"'] & Y \end{tikzcd}$$

with $\cup\!\!\mid$ (i.e. $\supseteq$) between $\gamma(Y^\sharp)$ and Y.

Pattern for the sound condition for each semantic operator $f^\sharp : A^\sharp \to B^\sharp$

$$f \circ \gamma_A \sqsubseteq_B \gamma_B \circ f^\sharp.$$

Then, Follows Sound Static Analysis

- In case $\mathbb{S}^\sharp$ is of finite-height and $F^\sharp$ is monotone or extensive, then

$$\bigsqcup_{i\geq 0} {F^\sharp}^i(\bot)$$

  is finitely computable and over-approximates the concrete semantics **lfp**F.
- Otherwise, find a widening operator $\bigtriangledown$, then the following chain $X_0 \sqsubseteq X_1 \sqsubseteq \cdots$

$$X_0 = \bot \qquad X_{i+1} = X_i \bigtriangledown F^\sharp(X_i)$$

  is finite and its last element over-approximates the concrete semantics **lfp**F.

Underlying Theorems (1/2)

Theorem (Sound static analysis by $F^\sharp$)

Given a program, let F and $F^\sharp$ be defined as in the framework. If $\mathbb{S}^\sharp$ is of finite-height (every chain $\mathbb{S}^\sharp$ is finite) and $F^\sharp$ is monotone or extensive, then

$$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)$$

is finitely computable and over-approximates **lfp**F*:*

$$\textbf{lfp}F \ \subseteq \ \gamma(\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)) \quad \text{or equivalently} \quad \alpha(\textbf{lfp}F) \ \sqsubseteq \ \bigsqcup_{i \geq 0} {F^\sharp}^i(\bot).$$

(Proof is in the supplementary note.)

Underlying Theorems (2/2)

Theorem (Sound static analysis by $F^\sharp$ and widening operator $\bigtriangledown$)

Given a program, let F and $F^\sharp$ be defined as in the framework. Let $\bigtriangledown$ be a widening operator. Then the following chain $Y_0 \sqsubseteq Y_1 \sqsubseteq \cdots$

$$Y_0 = \bot \qquad Y_{i+1} = Y_i \bigtriangledown F^\sharp(Y_i)$$

is finite and its last element $Y_{\lim}$ over-approximates $\mathbf{lfp}F$:

$$\mathbf{lfp}F \subseteq \gamma(Y_{\lim}) \quad \textit{or equivalently} \quad \alpha(\mathbf{lfp}F) \sqsubseteq Y_{\lim}.$$

(Proof is in the supplementary note.)

Definition (Widening operator)

A *widening* operator over an abstract domain $\mathbb{A}$ is a binary operator $\triangledown$, such that:

1. For all abstract elements a_0, a_1, we have

$$\gamma(a_0) \cup \gamma(a_1) \subseteq \gamma(a_0 \triangledown a_1)$$

2. For all sequence $(a_n)_{n\in\mathbb{N}}$ of abstract elements, the sequence $(a'_n)_{n\in\mathbb{N}}$ defined below is ultimately stationary:

$$\begin{cases} a'_0 & = & a_0 \\ a'_{n+1} & = & a'_n \triangledown a_n \end{cases}$$

Analysis Algorithm Based on Global Iterations: Basic Version (1/2)

- Case: $\mathbb{S}^\sharp$ is of finite-height and $F^\sharp$ is monotone or extensive
- Note the increasing chain

$$\bot \sqsubseteq (F^\sharp)^1(\bot) \sqsubseteq (F^\sharp)^2(\bot) \sqsubseteq \cdots$$

is finite and its biggest element is equal to

$$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot).$$

$\mathtt{C} \leftarrow \bot$
repeat
    $\mathtt{R} \leftarrow \mathtt{C}$
    $\mathtt{C} \leftarrow F^\sharp(\mathtt{C})$
until $\mathtt{C} \sqsubseteq \mathtt{R}$
return $\mathtt{R}$

Analysis Algorithm Based on Global Iterations: Basic Version (2/2)

- Case: $\mathbb{S}^\sharp$ is of infinite-height or $F^\sharp$ is neither monotonic nor extensive
- Use a widening operator $\bigtriangledown$

$$
\begin{array}{|l}
\mathtt{C} \leftarrow \bot \\
\text{repeat} \\
\qquad \mathtt{R} \leftarrow \mathtt{C} \\
\qquad \mathtt{C} \leftarrow \mathtt{C} \bigtriangledown F^\sharp(\mathtt{C}) \\
\text{until } \mathtt{C} \sqsubseteq \mathtt{R} \\
\text{return } \mathtt{R}
\end{array}
$$

Inefficiency of the Basic Algorithms

Recall the algirthm with $F^\sharp(\mathtt{C})$ being inlined:

$$
\begin{array}{|l}
\mathtt{C} \leftarrow \bot \\
\text{repeat} \\
\qquad \mathtt{R} \leftarrow \mathtt{C} \\
\qquad \mathtt{C} \leftarrow \mathtt{C} \bigtriangledown \underbrace{(\wp(\mathrm{id}, \sqcup) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp))}_{F^\sharp}(\mathtt{C}) \\
\text{until } \mathtt{C} \sqsubseteq \mathtt{R} \\
\text{return } \mathtt{R}
\end{array}
$$

- $|\mathtt{C}| \sim$ the number of labels in the input program!
- Better apply

$$\breve{\wp}(\hookrightarrow^\sharp)(\mathtt{C})$$

  only to necessary labels

Analysis Algorithm Based on Global Iterations: Worklist Version

- worklist: the set of labels whose input memories are changed in the previous iteration

$$
\begin{array}{|l}
\mathtt{C} : \mathbb{L} \to \mathbb{M}^\sharp \\
F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp) \\
\mathtt{WorkList} : \wp(\mathbb{L}) \\
\\
\mathtt{WorkList} \leftarrow \mathbb{L} \\
\mathtt{C} \leftarrow \bot \\
\text{repeat} \\
\quad\quad \mathtt{R} \leftarrow \mathtt{C} \\
\quad\quad \mathtt{C} \leftarrow \mathtt{C} \bigtriangledown F^\sharp(\mathtt{C}|_{\mathtt{WorkList}}) \\
\quad\quad \mathtt{WorkList} \leftarrow \{l \mid \mathtt{C}(l) \not\sqsubseteq \mathtt{R}(l), l \in \mathbb{L}\} \\
\text{until } \mathtt{WorkList} = \emptyset \\
\text{return } \mathtt{R}
\end{array}
$$

Improvement of the Worklist Algorithm

- Inefficient: $\texttt{WorkList} \leftarrow \{l \mid \texttt{C}(l) \not\sqsubseteq \texttt{R}(l), l \in \mathbb{L}\}$ re-scans all the labels.
  - Better: At application $\hookrightarrow^\sharp$ to $(l, \texttt{C}(l))$, if its result $(l', M^\sharp)$ is changed $(M^\sharp \not\sqsubseteq \texttt{C}(l'))$, add l' to the worklist.
- Inefficient: $\texttt{C} \bigtriangledown F^\sharp(C|_{\texttt{WorkList}})$ widens at all the labels.
  - Better: Apply $\bigtriangledown$ only at the target of a loop. Use $\cup^\sharp$ at other labels.

Summary: Recipe for Defining Sound Static Analysis(1/4)

1. Define $\mathbb{M}$ to be the set of memory states that can occur during program executions. Let $\mathbb{L}$ be the finite and fixed set of labels of a given program.
2. Define a concrete semantics as the **lfp**F where

$$
\begin{array}{rrcl}
\text{concrete domain} & \wp(\mathbb{S}) & = & \wp(\mathbb{L} \times \mathbb{M}) \\
\text{concrete semantic function} & F : \wp(\mathbb{S}) \to \wp(\mathbb{S}) & & \\
 & F(X) & = & I \cup \mathit{Step}(X) \\
 & \mathit{Step} & = & \breve{\wp}(\hookrightarrow) \\
 & \hookrightarrow & \subseteq & (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})
\end{array}
$$

The $\hookrightarrow$ is the one-step transition relation over $\mathbb{L} \times \mathbb{M}$.

Summary: Recipe for Defining Sound Static Analysis(2/4)

4. Define its abstract domain and abstract semantic function as

$$
\begin{array}{rrcl}
\text{abstract domain} & \mathbb{S}^\sharp & = & \mathbb{L} \to \mathbb{M}^\sharp \\
\text{abstract semantic function} & F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp & & \\
 & F^\sharp(X^\sharp) & = & \alpha(I) \cup^\sharp \mathit{Step}^\sharp(X^\sharp) \\
 & \mathit{Step}^\sharp & = & \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp) \\
 & \hookrightarrow^\sharp & \subseteq & (\mathbb{L} \times \mathbb{M}^\sharp) \times (\mathbb{L} \times \mathbb{M}^\sharp)
\end{array}
$$

The $\hookrightarrow^\sharp$ is the one-step abstract transition relation over $\mathbb{L} \times \mathbb{M}^\sharp$. Function π partitions a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp$ by the labels in $\mathbb{L}$ returning an element in $\mathbb{L} \to \wp(\mathbb{M}^\sharp)$ represented as a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp)$.

Summary: Recipe for Defining Sound Static Analysis(3/4)

5. Check the abstract domains $\mathbb{S}^\sharp$ and $\mathbb{M}^\sharp$ are CPOs, and forms a Galois-connection respectively with $\wp(\mathbb{S})$ and $\wp(\mathbb{M})$:

$$(\wp(\mathbb{S}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{S}^\sharp, \sqsubseteq) \quad \text{and} \quad (\wp(\mathbb{M}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

where the partial order $\sqsubseteq$ of $\mathbb{S}^\sharp$ is label-wise $\sqsubseteq_M$:

$$a^\sharp \sqsubseteq b^\sharp \quad \text{iff} \quad \forall l \in \mathbb{L} : a^\sharp(l) \sqsubseteq_M b^\sharp(l).$$

6. Check the abstract one-step transition $\hookrightarrow^\sharp$ and abstract union $\cup^\sharp$ satisfy:

$$\begin{array}{lcl} \breve{\wp}(\hookrightarrow) \circ \gamma & \subseteq & \gamma \circ \breve{\wp}(\hookrightarrow^\sharp) \\ \cup \circ (\gamma, \gamma) & \subseteq & \gamma \circ \cup^\sharp \end{array}$$

Summary: Recipe for Defining Sound Static Analysis(4/4)

7. Then, sound static analysis is defined as follows:
   - In case $\mathbb{S}^\sharp$ is of finite-height (every its chain is finite) and $F^\sharp$ is monotone or extensive, then

     $$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)$$

     is finitely computable and over-approximates the concrete semantics **lfp**F.
   - Otherwise, find a widening operator $\bigtriangledown$, then the following chain $X_0 \sqsubseteq X_1 \sqsubseteq \cdots$

     $$X_0 = \bot \qquad X_{i+1} = X_i \bigtriangledown F^\sharp(X_i)$$

     is finite and its last element over-approximates the concrete semantics **lfp**F.

Use Example: Target Language

$$
\begin{array}{lcll}
\mathtt{x} & \in & \mathbb{X} & \text{program variables} \\
C & ::= & & \text{statements} \\
 & | & \mathtt{skip} & \text{nop statement} \\
 & | & C\ ;\ C & \text{sequence of statements} \\
 & | & \mathtt{x} := E & \text{assignment} \\
 & | & \mathtt{input}(\mathtt{x}) & \text{read an integer input} \\
 & | & \mathtt{if}(B)\{C\}\mathtt{else}\{C\} & \text{condition statement} \\
 & | & \mathtt{while}(B)\{C\} & \text{loop statement} \\
 & | & \mathtt{goto}\ E & \text{goto with dynamically computed label} \\
E & ::= & & \text{expression} \\
 & | & n & \text{integer} \\
 & | & \mathtt{x} & \text{variable} \\
 & | & E + E & \text{addition} \\
B & ::= & & \text{boolean expression} \\
 & | & \mathtt{true} \mid \mathtt{false} & \\
 & | & E < E & \text{comparison} \\
 & | & E = E & \text{equality} \\
P & ::= & C & \text{program}
\end{array}
$$

Figure: Syntax of a simple imperative language

Use Example: Concrete State Transition Semantics

$$\mathsf{lfp} F$$

of the continuous function

$$\begin{array}{l} F : \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ F(X) = I \cup \mathit{Step}(X) \\ \mathit{Step}(X) = \breve{\wp}(\hookrightarrow). \end{array}$$

where

$$\mathbb{S} = \mathbb{L} \times \mathbb{M}$$

and

$$\begin{array}{llcl} \text{memories} & \mathbb{M} & = & \mathbb{X} \to \mathbb{V} \\ \text{values} & \mathbb{V} & = & \mathbb{Z} \cup \mathbb{L}. \end{array}$$

The state transition relation $(l, m) \hookrightarrow (l', m')$ is defined as follows.

$$\begin{array}{rcl} \texttt{skip} & : & (l,m) \hookrightarrow (\texttt{next}(l),\ m) \\ \texttt{input(x)} & : & (l,m) \hookrightarrow (\texttt{next}(l),\ \mathit{update}_{\texttt{x}}(m, z)) \qquad \text{for an input integer } z \\ \texttt{x} := E & : & (l,m) \hookrightarrow (\texttt{next}(l),\ \mathit{update}_{\texttt{x}}(m, \mathit{eval}_E(m))) \\ C_1; C_2 & : & (l,m) \hookrightarrow (\texttt{next}(l),\ m) \\ \texttt{if}(B)\{C_1\}\texttt{else}\{C_2\} & : & (l,m) \hookrightarrow (\texttt{nextTrue}(l),\ \mathit{filter}_B(m)) \\ & : & (l,m) \hookrightarrow (\texttt{nextFalse}(l),\ \mathit{filter}_{\neg B}(m)) \\ \texttt{while}(B)\{C\} & : & (l,m) \hookrightarrow (\texttt{nextTrue}(l),\ \mathit{filter}_B(m)) \\ & : & (l,m) \hookrightarrow (\texttt{nextFalse}(l),\ \mathit{filter}_{\neg B}(m)) \\ \texttt{goto}\ E & : & (l,m) \hookrightarrow (\mathit{eval}_E(m),\ m) \end{array}$$

Use Example: Abstract State

An abstract domain $\mathbb{M}^\sharp$ is a CPO such that

$$(\wp(\mathbb{M}), \subseteq) \xleftarrow[\alpha_M]{\gamma_M} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

defined as

$$M^\sharp \in \mathbb{M}^\sharp = \mathbb{X} \to \mathbb{V}^\sharp$$

where $\mathbb{V}^\sharp$ is an abstract domain that is a CPO such that

$$(\wp(\mathbb{V}), \subseteq) \xleftarrow[\alpha_V]{\gamma_V} (\mathbb{V}^\sharp, \sqsubseteq_V).$$

We design $\mathbb{V}^\sharp$ as

$$\mathbb{V}^\sharp = \mathbb{Z}^\sharp \times \mathbb{L}^\sharp$$

where $\mathbb{Z}^\sharp$ is a CPO that is Galois connected with $\wp(\mathbb{Z})$, and $\mathbb{L}^\sharp$ is the powerset $\wp(\mathbb{L})$ of labels.
All abstract domains are Galois-connected CPOs, homomorphic to their concrete correspondents.

Use Example: Abstract State Transition Semantics

Case the l-labeled statement of

$$
\begin{array}{rcl}
\texttt{skip} & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), M^\sharp) \\
\texttt{input(x)} & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), \textit{update}^\sharp_{\texttt{x}}(M^\sharp, \alpha(\mathbb{Z}))) \\
\texttt{x} := E & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), \textit{update}^\sharp_{\texttt{x}}(M^\sharp, \textit{eval}^\sharp_E(M^\sharp))) \\
C_1; C_2 & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), M^\sharp) \\
\texttt{if}(B)\{C_1\}\texttt{else}\{C_2\} & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{nextTrue}(l), \textit{filter}^\sharp{}_B(M^\sharp)) \\
 & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{nextFalse}(l), \textit{filter}^\sharp{}_{\neg B}(M^\sharp)) \\
\texttt{while}(B)\{C\} & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{nextTrue}(l), \textit{filter}^\sharp{}_B(M^\sharp)) \\
 & : & (l, M^\sharp) \hookrightarrow^\sharp (\texttt{nextFalse}(l), \textit{filter}^\sharp{}_{\neg B}(M^\sharp)) \\
\texttt{goto}\ E & : & (l, M^\sharp) \hookrightarrow^\sharp (l', M^\sharp) \quad \text{for } l' \in L \text{ of } (z^\sharp, L) = \textit{eval}^\sharp_E(M^\sharp)
\end{array}
$$

Let $F^\sharp$ be defined as the framework:

$$
\begin{array}{l}
F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp \\
F^\sharp(S^\sharp) = \alpha(I) \cup^\sharp \textit{Step}^\sharp(S^\sharp) \\
\textit{Step}^\sharp = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp).
\end{array}
$$

If the $\textit{Step}^\sharp$ and $\cup^\sharp$ are sound abstractions of, respectively, *Step* and $\cup$, as required by the framework:

$$
\begin{array}{rcl}
\breve{\wp}(\hookrightarrow) \circ \gamma & \subseteq & \gamma \circ \breve{\wp}(\hookrightarrow^\sharp) \\
\cup \circ (\gamma, \gamma) & \subseteq & \gamma \circ \cup^\sharp
\end{array}
$$

then we can use $F^\sharp$ to soundly approximates the concrete semantics $\mathbf{lfp}F$

Use Example: Defining Sound $\hookrightarrow^\sharp$

If each of the abstract semantic operators is a sound abstraction of its concrete correspondent, then $\hookrightarrow^\sharp$ is a sound abstraction of $\hookrightarrow$:

Theorem (Soundness of $\hookrightarrow^\sharp$)

If the semantic operators satisfy the following soundness properties:

$$
\begin{array}{rcl}
\wp(\mathit{eval}_E) \circ \gamma_M & \subseteq & \gamma_V \circ \mathit{eval}^\sharp_E \\
\wp(\mathit{update}_{\mathtt{x}}) \circ \times \circ (\gamma_M, \gamma_V) & \subseteq & \gamma_M \circ \mathit{update}^\sharp_{\mathtt{x}} \\
\wp(\mathit{filter}_B) \circ \gamma_M & \subseteq & \gamma_M \circ \mathit{filter}^\sharp_B \\
\wp(\mathit{filter}_{\neg B}) \circ \gamma_M & \subseteq & \gamma_M \circ \mathit{filter}^\sharp_{\neg B}
\end{array}
$$

then $\breve{\wp}(\hookrightarrow) \circ \gamma \;\sqsubseteq\; \gamma \circ \breve{\wp}(\hookrightarrow^\sharp)$. *(The $\times$ is the Cartesian product operator of two sets.)*

Use Example: Defining Sound $\cup^\sharp$

As of a sound $\cup^\sharp$, one candidate is the least upper bound operator $\sqcup$ if $\mathbb{S}^\sharp$ is closed by $\sqcup$, because

$$\begin{array}{rclr} (\gamma \circ \sqcup)(a^\sharp, b^\sharp) \ = \ \gamma(a^\sharp \sqcup b^\sharp) & \sqsupseteq & \gamma(a^\sharp) \cup \gamma(b^\sharp) & \text{by the monotonicity} \\ & = & (\cup \circ (\gamma, \gamma))(a^\sharp, b^\sharp). & \end{array}$$

Outline

Scalability Challenge

Figure: Call graph of `less-382` (23,822 lines of code)

Sparse Analysis

- Exploit the semantic sparsity of the input program to analyze
- Spatial sparsity & temporal sparsity

Right part at right moment

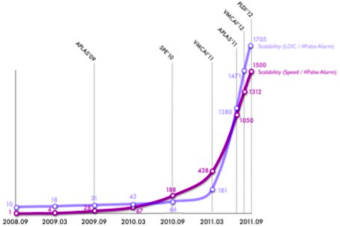
Example Performance Gain by Sparse Analysis

- Sparrow: a "sound", global C analyzer for the memory safety property (no overrun, no null-pointer dereference, etc.)

  `http://github.com/ropas/sparrow`

- ~ 10 hours in analyzing million lines of C

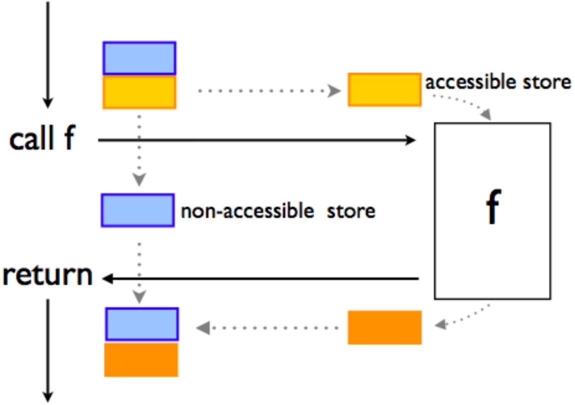
Sparrow The Early Bird — sound-&-global version



- < 1.4M in 10hrs with intervals
- < 0.14M in 20hrs with octagons

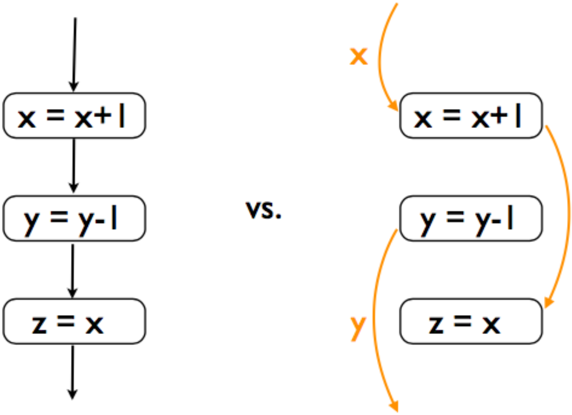
Spatial Sparcity

Each program portion accesses only a small part of the memory.



Temporal Sparcity

After the def of a memory, its use is far.

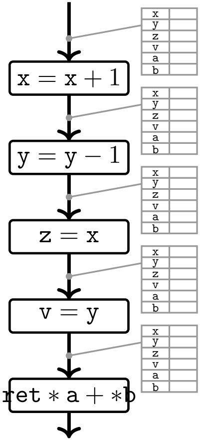


Example (Code fragment)

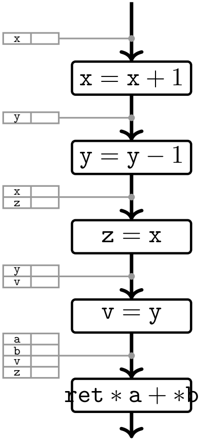
```
x = x + 1;
y = y - 1;
z = x;
v = y;
ret *a + *b
```

Assume that `a` points to `v` and `b` to `z`.

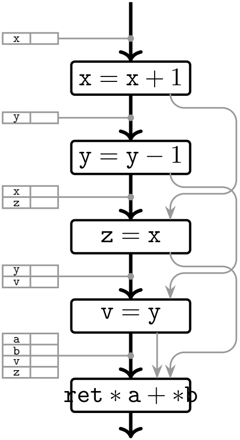
Spatial and Temporal Sparsity of the Example Code



(a) Without exploiting the sparsities



(b) Spatial sparsity



(c) Spatial & temporal sparsity

Exploiting Spatial Sparsity: Need $Access^\sharp(l)$

"abstract garbage collecition", "frame rule"

$$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$$

becomes

$$F^\sharp_{sparse} : (\mathbb{L} \to \mathbb{M}^\sharp_{sparse}) \to (\mathbb{L} \to \mathbb{M}^\sharp_{sparse})$$

where

$$\mathbb{M}^\sharp_{sparse} = \{M^\sharp \in \mathbb{M}^\sharp \mid dom(M^\sharp) = Access^\sharp(l), l \in \mathbb{L}\} \cup \{\bot\}.$$

Exploiting Temporal Sparsity: Need Def-Use Chain

Need the def-use chain information as follows.

- we streamline the abstract one-step relation

$$(l, M^\sharp) \hookrightarrow^\sharp (l', M'^\sharp) \quad \text{for } l' \in \mathtt{next}^\sharp(l, M^\sharp).$$

  so that the link $\hookrightarrow^\sharp$ should follow the **def-use chain**:
  - from (def) a label where a location is defined
  - to (use) a label where the defined location is read

Precision Preserving Sparse Analysis Framework

Goal

$$F^\sharp : D^\sharp \to D^\sharp \quad \overset{\text{sparsify}}{\Longrightarrow} \quad F^\sharp_{sparse} : D^\sharp \to D^\sharp$$

$$\textsf{lfp} F^\sharp \quad \overset{\text{still}}{=} \quad \textsf{lfp} F^\sharp_{sparse}$$

Precision Preserving Sparse Analysis: for Spatial Sparsity (1/3)

Need to safely estimate

$$Access^\sharp(l).$$

Use yet another sound static analysis, a futher abstraction:

$$(\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

(a "flow-insensitive" version of the "flow-sensitive" analysis design)

Precision Preserving Sparse Analysis: for Temporal Sparsity (2/3)

- Let

$$D^\sharp : \mathbb{L} \rightarrow \wp(\mathbb{X}) \text{ and } U^\sharp : \mathbb{L} \rightarrow \wp(\mathbb{X})$$

  be the def and use sets from the original analysis.
- Need to safely estimate $D^\sharp$ and $U^\sharp$.
- Use yet another sound static analysis to compute

$$D^\sharp_{pre} \text{ and } U^\sharp_{pre}$$

  such that
  - $\forall l \in \mathbb{L} : D^\sharp_{pre}(l) \supseteq D^\sharp(l) \quad \text{and} \quad U^\sharp_{pre}(l) \supseteq U^\sharp(l)$.
  - $\forall l \in \mathbb{L} : U^\sharp_{pre}(l) \supseteq D^\sharp_{pre}(l) \setminus D^\sharp(l)$.

Precision Preserving Sparse Analysis: for Temporal Sparsity (3/3)

Let $D^\sharp_{pre}$ and $U^\sharp_{pre}$ be, respectively, safe def and use sets from a pre-analysis as defined before.

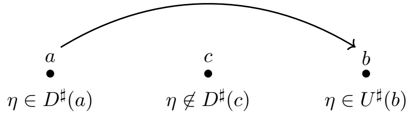
Definition (Precision preserving def-use chain)

Label a to label b is a def-use chain for an abstract location η whenever $\eta \in D^\sharp_{pre}(a)$, $\eta \in U^\sharp_{pre}(b)$, and η may not be re-defined inbetween the two labels.

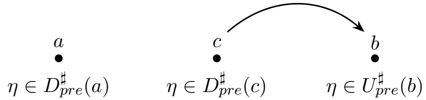
Precision preservation

Then, the resulting sparse analysis version has the same precision as the original non-sparse analysis.

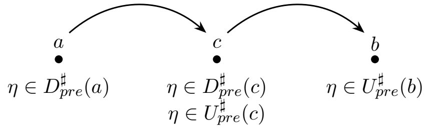
Need for the Second Condition for $D^\sharp_{pre}$ and $U^\sharp_{pre}$



(d) Original analysis def-use edge for η



(e) Missing def-use edge (a to b) for η because of over-approximate $D^\sharp_{pre}(c)$



(f) Recovered def-use edge (a to b via c) for η by safe $U^\sharp_{pre}(c)$

Outline

Specialized Frameworks

Practical altenatives to the aforementioned general, abstract interpretation framework

- for simple languages and properties,
- $\exists$frameworks that are simple yet powerful enough
- review of their limitations

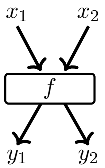
Three specialized frameworks:

- static analysis by equations
- static analysis by monotonic closure
- static analysis by proof construction

Static Analysis by Equations

- Static analysis = equation setup and resolution
  - equations capture all the executions of the program
  - a solution of the equations is the analysis result
- Represent programs by control-flow graphs
  - nodes for semantic functions (statements)
  - edges for control flow
- Straightforward to set up sound equations

For each node



we set up equations

$$y_1 = f(x_1 \sqcup x_2)$$
$$y_2 = f(x_1 \sqcup x_2)$$

Example: Data-Flow Analysis for Integer Intervals

Example (Data-flow analysis)

```
input (x);
while (x <= 99)
    x := x+1
```

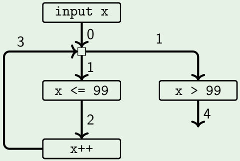


Figure: Control-flow graph

$$
\begin{aligned}
x_0 &= [-\infty, +\infty] \\
x_1 &= x_0 \sqcup x_3 \\
x_2 &= x_1 \sqcap [-\infty, 99] \\
x_3 &= x_2 \oplus 1 \\
x_4 &= x_1 \sqcap [100, +\infty]
\end{aligned}
$$

Figure: A set of equations for the program

Limitations

Not powerful enough for arbitrary languages

- control-flow before analysis?
  - control is also computed in modern languages
  - no: the dichotomy of control being fixed and data being dynamic
- sound transformation function?
  - error prone for complicated features of modern languages
  - e.g. function call/return, function as a data, dynamic method dispatch, exception, pointer manipulation, dynamic memory allocation, ...
- lacks a systematic approach
  - to prove the correctness of the analysis
  - to vary the accuracy of the analysis

Static Analysis by Monotonic Closure (1/2)

- Static analysis = setting up initial facts then collecting new facts by a kind of chain reaction
  - has rules for collecting initial facts
  - has rules for generating new facts from existing facts
- the initial facts immediate from the program text
- the chain reaction steps simulate the program semantics
- the universe of facts are finite for each program
- analysis accumulates facts until no more possible

Static Analysis by Monotonic Closure (2/2)

- let R be the set of the chain-reaction rules
- let X_0 be the initial fact set
- let *Facts* be the set of all possible facts

Then, the analysis result is

$$\bigcup_{i \geq 0} Y_i,$$

where

$$\begin{array}{rcl} Y_0 & = & X_0, \\ Y_{i+1} & = & Y \text{ such that } Y_i \vdash_R Y. \end{array}$$

Or, equivalently, the analysis result is the least fixpoint

$$\bigcup_{i \geq 0} \phi^i(\emptyset)$$

of monotonic function $\phi : \wp(\mathit{Facts}) \rightarrow \wp(\mathit{Facts})$:

$$\phi(X) = X_0 \cup (Y \text{ such that } X \vdash_R Y).$$

Example: Pointer Analysis (1/3)

$$
\begin{array}{rcll}
P & ::= & C & \text{program} \\
C & ::= & & \text{statement} \\
 & | & L \ := \ R & \text{assignment} \\
 & | & C \ ; \ C & \text{sequence} \\
 & | & \texttt{while}\ B\ C & \text{while-loop} \\
L & ::= & x \mid *x & \text{target to assign to} \\
R & ::= & n \mid x \mid *x \mid \&x & \text{value to assign} \\
B & & & \text{Boolean expression}
\end{array}
$$

- Goal: estimate all "points-to" relations between variables that can occur during executions
- $\texttt{a} \rightarrow \texttt{b}$: variable a can point to (can have the address of) variable b

Example: Pointer Analysis (2/3)

The initial facts that are obvious from the program text are collected by this rule:

$$\frac{x \ := \ \&y}{x \rightarrow y}$$

The chain-reaction rules are as follows for other cases of assignments:

$$\frac{x \ := \ y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{x \ := \ *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w}$$

$$\frac{*x \ := \ y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \frac{*x \ := \ *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}$$

$$\frac{*x \ := \ \&y \quad x \rightarrow w}{w \rightarrow y}$$

Example: Pointer Analysis (3/3)

Example (Pointer analysis steps)

```
x := &a ; y := &x ;
while B
    *y := &b ;
*x := *y
```

- Initial facts are from the first two assignments:

$$\mathtt{x} \rightarrow \mathtt{a}, \ \ \mathtt{y} \rightarrow \mathtt{x}$$

- From $\mathtt{y} \rightarrow \mathtt{x}$ and the while-loop body, add

$$\mathtt{x} \rightarrow \mathtt{b}$$

- From the last assignment:
  - from $\mathtt{x} \rightarrow \mathtt{a}$ and $\mathtt{y} \rightarrow \mathtt{x}$, add $\mathtt{a} \rightarrow \mathtt{a}$
  - from $\mathtt{x} \rightarrow \mathtt{b}$ and $\mathtt{y} \rightarrow \mathtt{x}$, add $\mathtt{b} \rightarrow \mathtt{b}$
  - from $\mathtt{x} \rightarrow \mathtt{a}$, $\mathtt{y} \rightarrow \mathtt{x}$, and $\mathtt{x} \rightarrow \mathtt{b}$, add $\mathtt{a} \rightarrow \mathtt{b}$
  - froom $\mathtt{x} \rightarrow \mathtt{b}$, $\mathtt{y} \rightarrow \mathtt{x}$, and $\mathtt{x} \rightarrow \mathtt{a}$, add $\mathtt{b} \rightarrow \mathtt{a}$

Limitations

Not powerful enough for arbitrary language

- sound rules?
  - error prone for complicated features of modern languages
  - e.g. function call/return, function as a data, dynamic method dispatch, exception, pointer manipulation, dynamic memory allocation, ...
- accuracy problem
  - consider program a set of statements, with no order between them
  - rules do not consider the control flow
  - the analysis blindly collects every possible facts when rules hold
  - accuracy improvement by more elaborate rules, but no systematic way for soundness proof

Static Analysis by Proof Construction

- Static analysis = proof construction in a finite proof system
- finite proof system = a finite set of inference rules for a predefined set of judgments
- The soundness corresponds to the soundness of the proof system.
  - the input program is provable $\Rightarrow$ the program satisfies the proven judgment.

Example: Type Inference (1/4)

$$
\begin{array}{rcll}
P & ::= & E & \text{program} \\
E & ::= & & \text{expression} \\
 & \mid & n & \text{integer} \\
 & \mid & \mathtt{x} & \text{variable} \\
 & \mid & \lambda \mathtt{x}.E & \text{function} \\
 & \mid & E\ E & \text{function application}
\end{array}
$$

- judgment that says expression E has type τ is written as

$$\Gamma \vdash E : \tau$$

- Γ is a set of type assumptions for the free variables in E.

Example: Type Inference (2/4)

Consider *simple types*

$$\tau ::= int \mid \tau \rightarrow \tau$$

$$\frac{}{\Gamma \vdash n : int} \qquad \frac{\mathtt{x} : \tau \ \in \Gamma}{\Gamma \vdash \mathtt{x} : \tau}$$

$$\frac{\Gamma + \mathtt{x} : \tau_1 \vdash E : \tau_2}{\Gamma \vdash \lambda \mathtt{x}.E : \tau_1 \rightarrow \tau_2} \qquad \frac{\Gamma \vdash E_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash E_1\ E_2 : \tau_2}$$

Figure: Proof rules of simple types

Theorem (Soundness of the proof rules)

Let E be a program, an expression without free variables. If $\emptyset \vdash E : \tau$, then the program runs without a type error and returns a value of type τ if it terminates.

Example: Type Inference (3/4)

Program

$$(\lambda \mathrm{x}.\mathrm{x}\ 1)(\lambda \mathrm{y}.\mathrm{y})$$

is typed int because we can prove

$$\emptyset \vdash (\lambda \mathrm{x}.\mathrm{x}\ 1)(\lambda \mathrm{y}.\mathrm{y}) : int$$

as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{x} : int \to int \ \in \{\mathtt{x} : int \to int\}}{\{\mathtt{x} : int \to int\} \vdash \mathtt{x} : int \to int} \quad \dfrac{}{\{\mathtt{x} : int \to int\} \vdash 1 : int}}{\{\mathtt{x} : int \to int\} \vdash \mathtt{x}\ 1 : int}}{\emptyset \vdash \lambda \mathtt{x}.\mathtt{x}\ 1 : (int \to int) \to int} \quad \dfrac{\dfrac{\mathtt{y} : int \ \in \{\mathtt{y} : int\}}{\{\mathtt{y} : int\} \vdash \mathtt{y} : int}}{\emptyset \vdash \lambda \mathtt{y}.\mathtt{y} : int \to int}}{\emptyset \vdash (\lambda \mathrm{x}.\mathrm{x}\ 1)(\lambda \mathrm{y}.\mathrm{y}) : int}$$

Example: Type Inference (4/4)

Algorithm

- given a program E, $V(\emptyset, E, \alpha)$ returns type equations.

$$
\begin{array}{rcll}
V(\Gamma, n, \tau) & = & \{\tau \doteq int\} & \\
V(\Gamma, \mathtt{x}, \tau) & = & \{\tau \doteq \Gamma(\mathtt{x})\} & \\
V(\Gamma, \lambda \mathtt{x}.E, \tau) & = & \{\tau \doteq \alpha_1 \to \alpha_2\} \cup V(\Gamma + \mathtt{x} : \alpha_1, E, \alpha_2) & (\text{new } \alpha_i) \\
V(\Gamma, E_1\ E_2, \tau) & = & V(\Gamma, E_1, \alpha \to \tau) \cup V(\Gamma, E_2, \alpha) & (\text{new } \alpha)
\end{array}
$$

- solving the equations is done by the *unification* procedure

Theorem (Correctness of the algorithm)

Solving the equations $\equiv$ *proving in the simple type system*

More precise analysis?

- need new sound proof rules (e.g.,*polymorphic type systems*)

Limitations

- For target languages that lack a sound static type system, we have to invent it.
  - design a finite proof system
  - prove the soundness of the proof system
  - design its algorithm that automates proving
  - prove the correctness of the algorithm
- What if the unification procedure is not enough?
  - for some properties, the algorithm can generate constraints that are unsolvable by the unification procedure
- For some conventional imperative languages, sound and precise-enough static type systems are elusive.

Static Analysis: an Abstract Interpretation Perspective

Thank you!