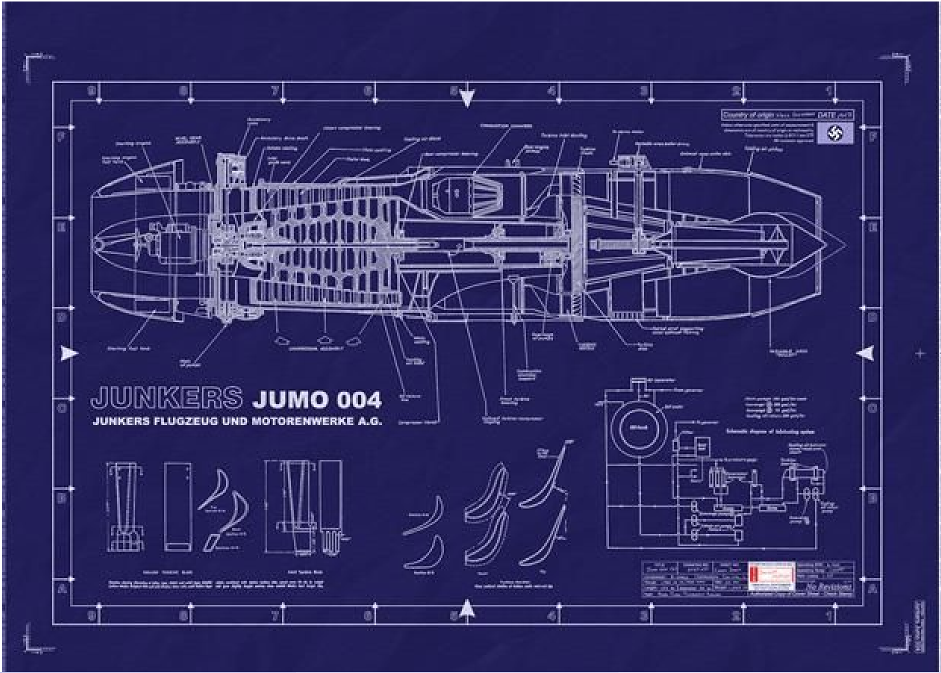
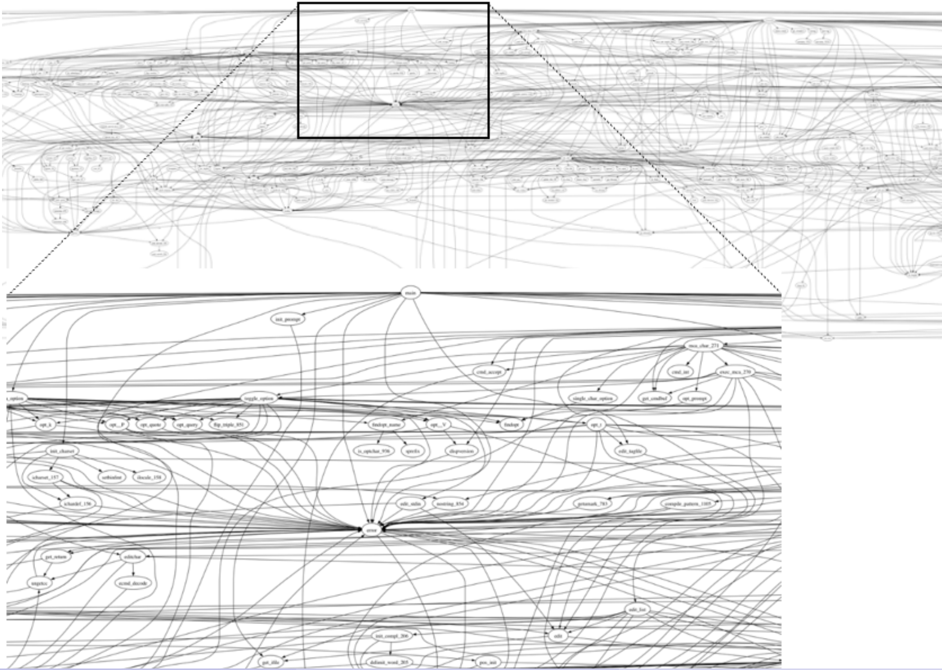
Static Program Analysis Overview

Kwangkeun Yi

Seoul National University, Korea

Outline


JUNKERS JUMO 004
JUNKERS FLUGZEUG UND MOTORENWERKE A.G.


main
init_prompt
toggle_option
error
edit

The Common Goal

	Computing area	Other engineering areas
Object	Software	Machine/building/circuit/chemical cess design
Execution subject	Computer runs it	Nature runs it
Our question	Will it work as intended?	Will it work as intended?
Our knowledge	Program analysis	Newtonian mechanics, Maxwell e tions, Navier-Stokes equations, the dynamic equations, and other princi

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:

- no crashing error
  - no divide by zero, no bus error in C, no uncaught exceptions
- no invariant violation
  - some data structure should never get broken
- no value overrun
  - a variable's values 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**
- Software has $\sim \infty$ states: the models need **approximation or non-termination (semi-algorithm)**

Conservative Static Analysis

Approach

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:
  - optimizing compilers relies on it (supposed to)
  - Astrée, Sparrow, Facebook Infer, ML type systems, ...
- **Automatic**
- Incompleteness: **may reject safe programs** (**false alarms**)
- Analysis algorithms **reason over program semantics**

Bug Finding

Approach

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

- commercial tools: 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/no	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** (software)
- **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 $\text{Exact}(P)$ of $[\![P]\!]$ is **impossible, in general**.

  For a Turing-complete language $\mathbb{L}$,
  $\not\exists$algorithm Exact : $\text{Exact}(P) = [\![P]\!]$ for all P in $\mathbb{L}$.

- Otherwise, we can solve the Halting Problem.
  - Given P, see if $\text{Exact}(P; 1/0)$ 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
```

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

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

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

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

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

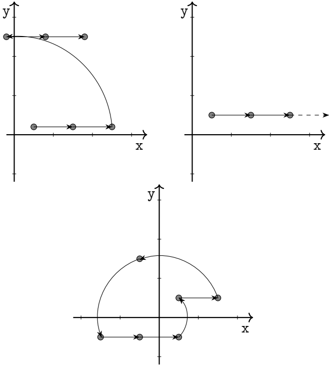
Outline

Example Language

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

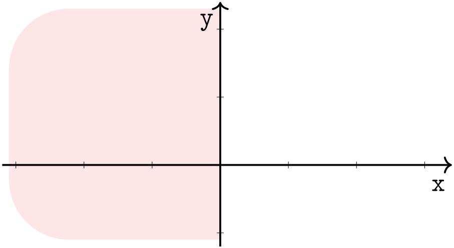
Example (Semantics)

$$
\begin{array}{l}
\texttt{init}([0,1]\times[0,1]);\\
\texttt{translation}(1,0);\\
\texttt{iter}\{\\
\quad\{\\
\qquad\texttt{translation}(1,0)\\
\quad\}\texttt{or}\{\\
\qquad\texttt{rotation}(0,0,90^\circ)\\
\quad\}\\
\}
\end{array}
$$

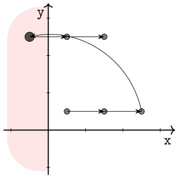


Analysis Goal Is Safety Property: Reachability

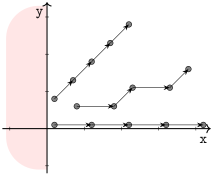
Analyze the set of reachable points, to check if the set intersects with a no-fly zone. Suppose that the no-fly zone is:



Correct or Incorrect Executions



(a) An incorrect execution

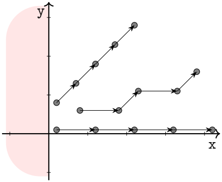


(b) Correct executions

An Example Safe Program

Example

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



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

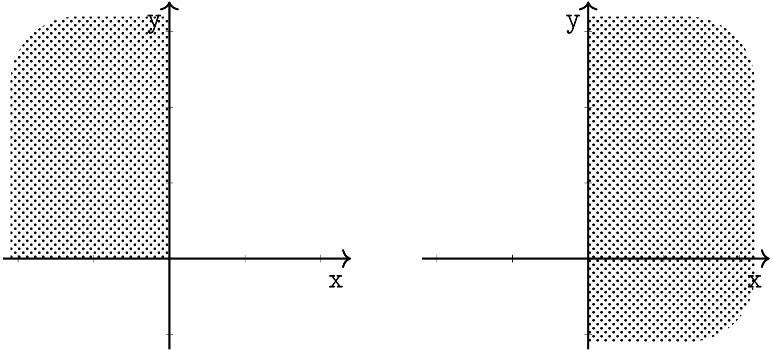
Definition (Abstraction)

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

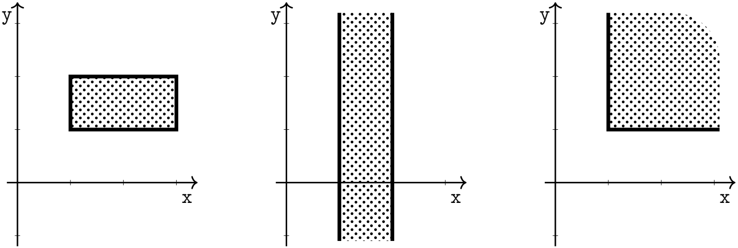


(c) Concretization of $[\mathtt{x} \leq 0, \mathtt{y} \geq 0]$ (d) Concretization of $[\mathtt{x} \geq 0]$

Figure: Signs abstraction

Abstraction Example 2: Interval Abstraction

The abstract elements: conjunctions of non-relational inequality constraints: $c_1 \leq \mathtt{x} \leq c_2$, $c'_1 \leq \mathtt{y} \leq c'_2$



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

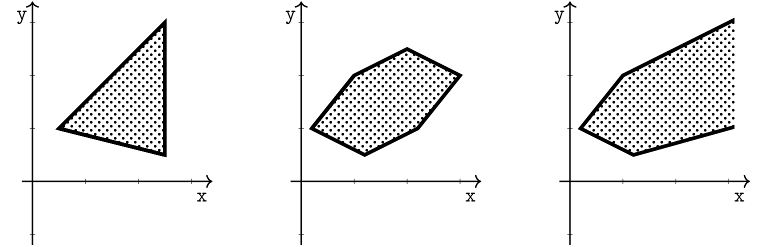
(b) Concretization of $[1 \leq \mathtt{x} \leq 2]$

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

Figure: Intervals abstraction

Abstraction Example 3: Convex Polyhedra Abstraction

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



(a) Concretization of a_0

(b) Concretization of a_1

(c) Concretization of a_2

Figure: Convex polyhedra abstraction

An Example Program, Again

Example

$\mathtt{init}([0,1] \times [0,1]);$
$\mathtt{iter}\{$
$\quad\{$
$\qquad\mathtt{translation}(1,0)$
$\quad\}\mathtt{or}\{$
$\qquad\mathtt{translation}(0.5,0.5)$
$\quad\}$
$\}$

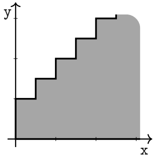
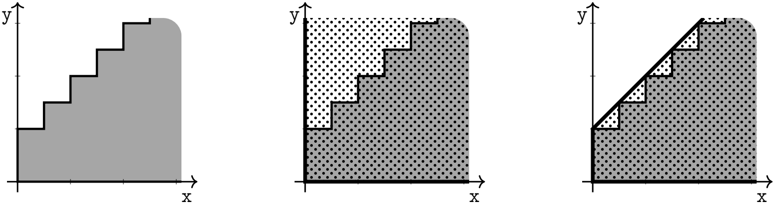


Figure: Reachable states

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**.

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

Sound Analysis Function as a Diagram

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

Figure: Sound analysis of a program p

Abstract Semantics Computation

Recall the example language

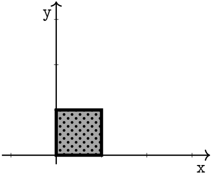
p	::=	init($\mathfrak{R}$)	initialization, with a state in $\mathfrak{R}$
	$\mid$	translation(u, v)	translation by vector (u, v)
	$\mid$	rotation(u, v, θ)	rotation defined by center (u, v) and angle θ
	$\mid$	p ; p	sequence of operations
	$\mid$	{p}or{p}	non-deterministic choice
	$\mid$	iter{p}	non-deterministic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

Approach

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

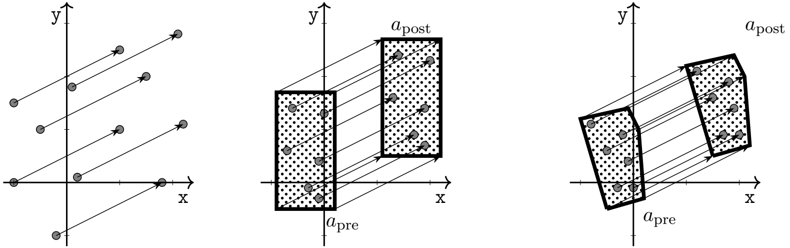
Abstract Semantics Computation: $\texttt{init}(\mathfrak{R})$

- 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 $\texttt{init}([0,1] \times [0,1])$ is



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

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



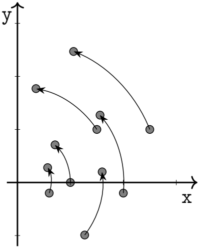
(a) Concrete semantics

(b) Intervals

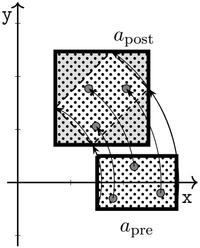
(c) Convex polyhedra

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

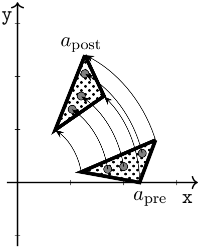
Abstract Semantics Computation: $\texttt{rotation}(u, v, \theta)$



(d) Concrete semantics



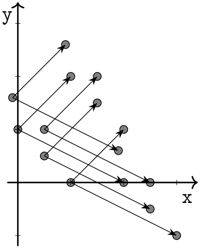
(e) Intervals



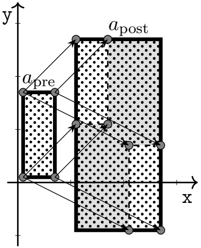
(f) Convex polyhedra

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

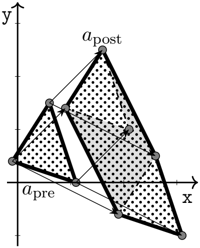
Abstract Semantics Computation: $\{\mathtt{p}\}\mathtt{or}\{\mathtt{p}\}$



(g) Concrete semantics



(h) Intervals



(i) Convex polyhedra

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

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

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

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

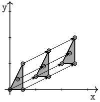
$\texttt{iter}\{\texttt{p}\}$ is equivalent to

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

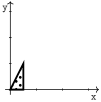
Abstract Semantics Computation: iter{p} (2/5)

Example (Abstract iteration)

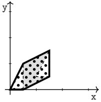
$\texttt{init}(\{(\mathtt{x},\mathtt{y}) \mid 0 \leq \mathtt{y} \leq 2\mathtt{x} \text{ and } \mathtt{x} \leq 0.5\});$
$\texttt{iter}\{$
$\quad \texttt{translation}(1, 0.5)$
$\}$



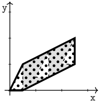
(j) Concrete semantics



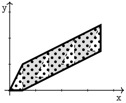
(k) Analysis of p_0 (0 iteration)



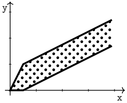
(l) Analysis of p_1 (up to 1 iteration)



(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: iter{p} (3/5)

Recall

$$\begin{aligned} \texttt{iter}\{\texttt{p}\} &= \{\}\ \texttt{or}\ \{\texttt{p}\}\ \texttt{or}\ \{\texttt{p};\texttt{p}\}\ \texttt{or}\cdots \\ &= \lim_i \texttt{p}_i \end{aligned}$$

where

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

Hence,

$$\texttt{analysis}(\texttt{iter}\{\texttt{p}\}, a) = \begin{cases} \texttt{R} \leftarrow a; \\ \text{repeat} \\ \qquad \texttt{T} \leftarrow \texttt{R}; \\ \qquad \texttt{R} \leftarrow \texttt{widen}(\texttt{R}, \texttt{analysis}(\texttt{p}, \texttt{R})); \\ \text{until } \texttt{inclusion}(\texttt{R}, \texttt{T}) \\ \text{return } \texttt{T}; \end{cases}$$

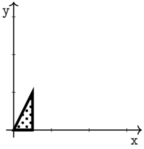
$$\text{operator } \texttt{widen} \quad \begin{cases} \text{over approximates unions} \\ \text{enforces finite convergence} \end{cases}$$

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

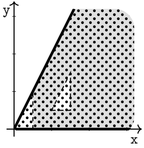
Example (Abstract iteration with widening)

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

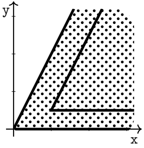
- The constraints $0 \le \mathrm{y}$ and $\mathrm{y} \le 2\mathrm{x}$ are stable after iteration 1; thus, they are preserved.
- The constraint $\mathrm{x} \le 0.5$ is not preserved; thus, it is discarded.



(a) Iteration 0



(b) Iteration 1



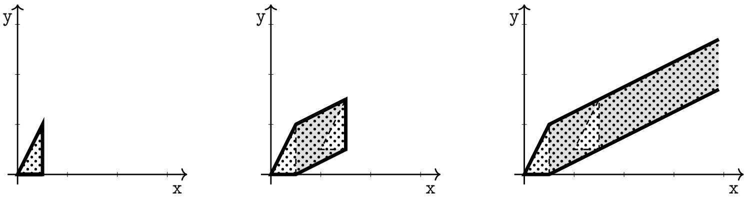
(c) Iteration 2 and limit

Figure: Abstract iteration with widening

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

Example (Loop unrolling)

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



(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.

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

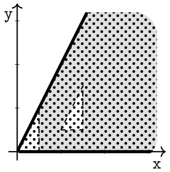
Sound analysis

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 $(\mathrm{x}, \mathrm{y}) \in \gamma(a)$,

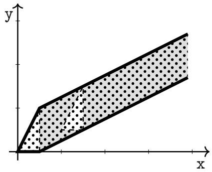
$$(\mathrm{x}', \mathrm{y}') \in \gamma(\texttt{analysis}(\mathrm{p}, a))$$

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`$(\mathtt{p}, -)$ 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 $\mathtt{p}$ is verified. Otherwise, we don't know.**



(a) An example $\mathfrak{R}$



(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

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