

Static Program Analysis Overview

Kwangkeun Yi

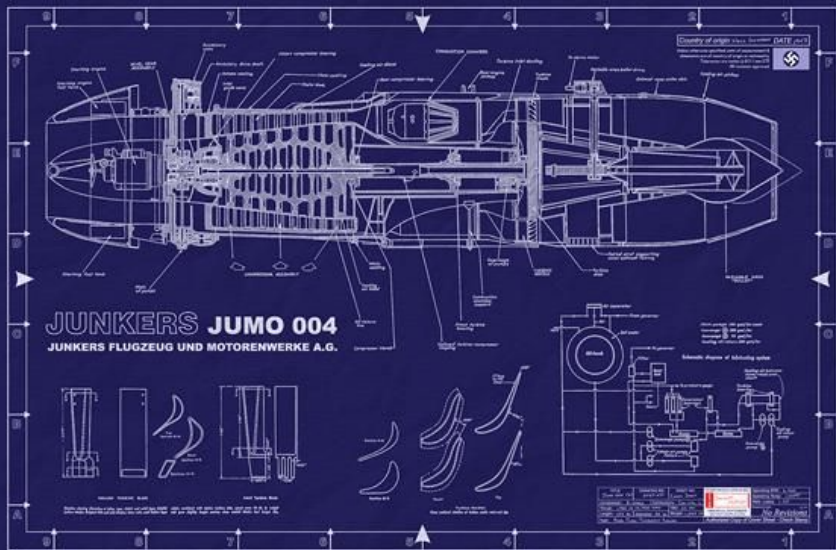
Seoul National University, Korea

1 Introduction

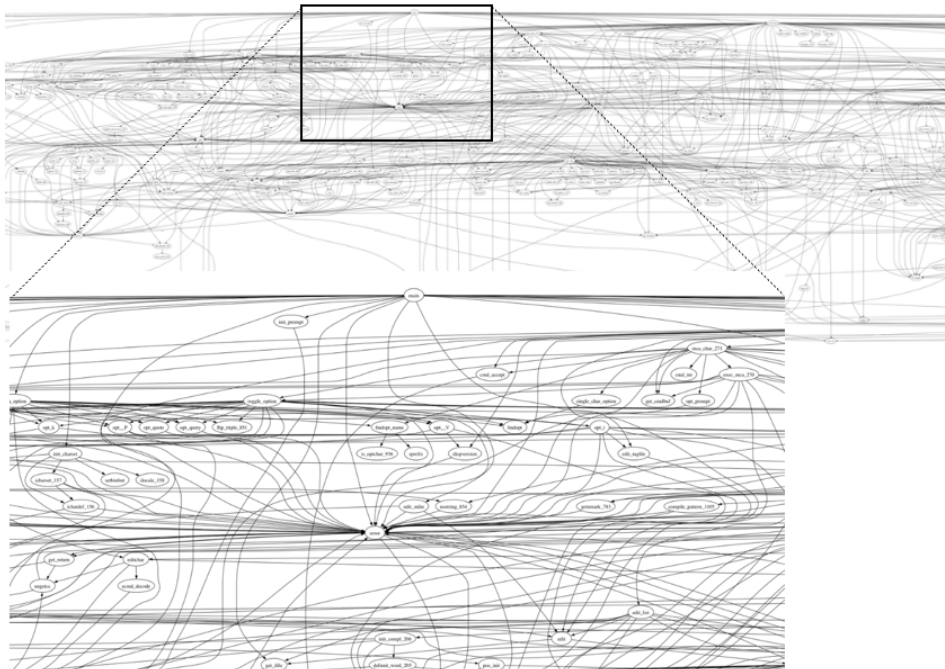
2 Static Analysis: a Gentle Introduction

Outline

- 1 Introduction
- 2 Static Analysis: a Gentle Introduction



Introduction



The Common Goal

	Computing area	Other engineering areas
Object	Software	Machine/building/circuit/chemical process 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 equations, Navier-Stokes equations, the dynamic equations, and other principles

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 $\llbracket P \rrbracket \subseteq \mathcal{S}$ where

- **the semantics** $\llbracket P \rrbracket$ = the set of all behaviors of P
- **the specification** \mathcal{S} = the set of acceptable behaviors

Semantics $\llbracket P \rrbracket$ and Semantic Properties \mathcal{S}

Semantics $\llbracket P \rrbracket$:

- compositional style (“denotational”)
 - ▶ $\llbracket AB \rrbracket = \dots \llbracket A \rrbracket \dots \llbracket B \rrbracket \dots$
- transitional style (“operational”)
 - ▶ $\llbracket AB \rrbracket = \{s_0 \hookrightarrow s_1 \hookrightarrow \dots, \dots\}$

Semantic properties \mathcal{S} :

- safety 늘 유지되는 성질, 내내성질, 늘성질
 - ▶ some behavior observable in *finite* time will never occur.
- liveness 언젠가는 일어날 성질, 결국성질, 언젠간성질
 - ▶ 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:** $\text{analysis}(P) = \text{yes} \implies P$ satisfies the specification
- **Completeness:** $\text{analysis}(P) = \text{yes} \iff P$ 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 $\llbracket P \rrbracket \subseteq \mathcal{S}$,
- **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 $\llbracket P \rrbracket$ Beforehand

- Finite & exact computation $\text{Exact}(P)$ of $\llbracket P \rrbracket$ is **impossible, in general**.

For a Turing-complete language \mathbb{L} ,
 \nexists algorithm $\text{Exact} : \text{Exact}(P) = \llbracket P \rrbracket$ 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** $\llbracket P \rrbracket^\sharp$ of $\llbracket P \rrbracket$

- “finite”, hence
 - ▶ automatic (algorithm) &
 - ▶ static (without executing P)
- “sound”
 - ▶ over-approximation of $\llbracket P \rrbracket$

Hence, ushers us to sound analysis:

$$(\text{analysis}(P) = \text{check } \llbracket P \rrbracket^\sharp \subseteq \mathcal{S}) \implies (P \text{ satisfies property } \mathcal{S})$$

Need Formal Frameworks of Static Analysis (1/2)

Suppose that

- We are interested in the value ranges of variables.
- How to finitely estimate $\llbracket P \rrbracket$ 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.

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

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
```

how?
 \implies

```
x1 = [-∞, +∞] or x3
x2 = x1 and [-∞, 99]
x3 = x2 + 1
x4 = x1 and [100, +∞]
```

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

```
x1 = [-∞, +∞] or x3
x2 = x1 and [-∞, 99]
x3 = x2 + 1
x4 = x1 and [100, ∞]
```

how?
 \implies

```
x1 = [-∞, +∞]
x2 = [-∞, 99]
x3 = [-∞, 100]
x4 = [100, +∞]
```

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

Outline

1 Introduction

2 Static Analysis: a Gentle Introduction

Example Language

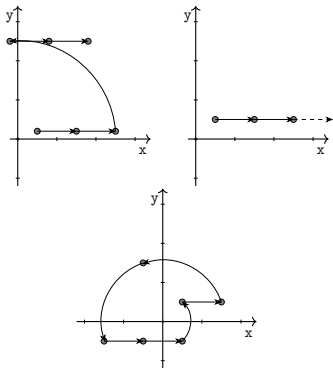
$p ::=$	$\text{init}(\mathfrak{R})$	initialization, with a state in \mathfrak{R}
	$\text{translation}(u, v)$	translation by vector (u, v)
	$\text{rotation}(u, v, \theta)$	rotation by center (u, v) and angle θ
	$p ; p$	sequence of operations
	$\{p\} \text{or} \{p\}$	non-deterministic choice
	$\text{iter}\{p\}$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

Example (Semantics)

```

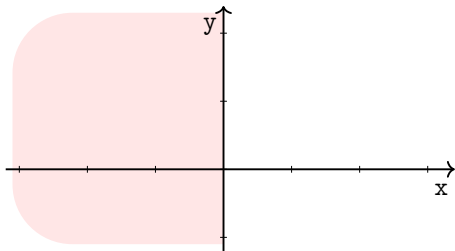
init([0,1] × [0,1]);
translation(1,0);
iter{
  {
    translation(1,0)
  }or{
    rotation(0,0,90°)
  }
}

```

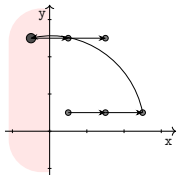


Analysis Goal Is Safety Property: Reachability

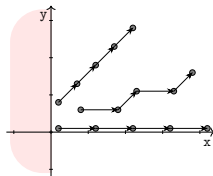
Analyze the set of reachable points, to check if the set intersects with a no-fly zone. Suppose that the no-fly zone is:



Correct or Incorrect Executions



(a) An incorrect execution

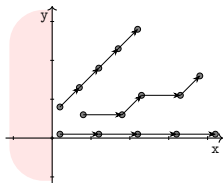


(b) Correct executions

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

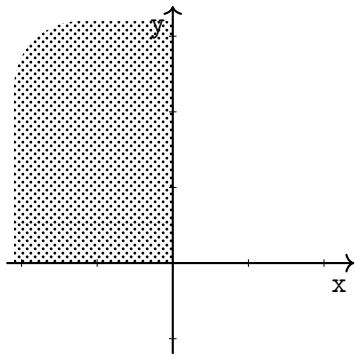
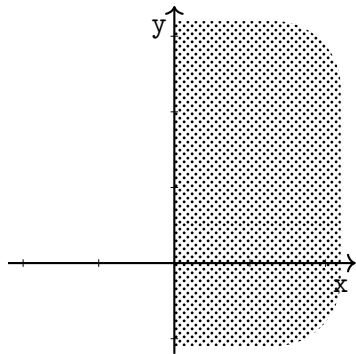
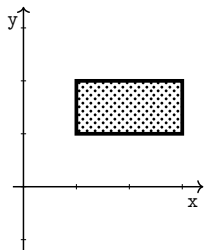
(c) Concretization of $[x \leq 0, y \geq 0]$ (d) Concretization of $[x \geq 0]$

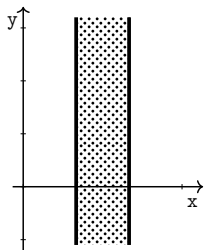
Figure: Signs abstraction

Abstraction Example 2: Interval Abstraction

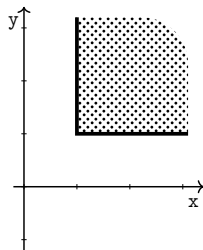
The abstract elements: conjunctions of non-relational inequality constraints:
 $c_1 \leq x \leq c_2, c'_1 \leq y \leq c'_2$



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



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



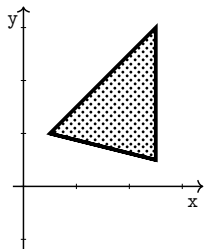
(c) Concretization of
 $[1 \leq x, 1 \leq y]$

Figure: Intervals abstraction

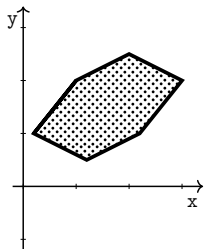
Abstraction Example 3: Convex Polyhedra Abstraction

The abstract elements: conjunctions of linear inequality constraints:

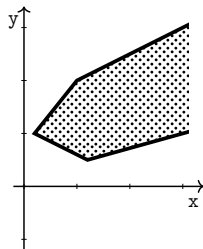
$$c_1x + c_2y \leq c_3$$



(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)  
  }  
}
```

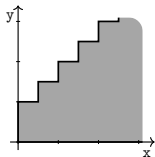
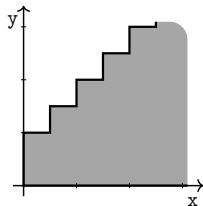
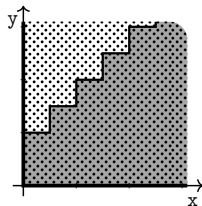


Figure: Reachable states

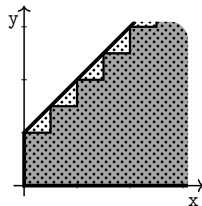
Abstractions of the Semantics of the Example Program



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

Sound Analysis Function as a Diagram

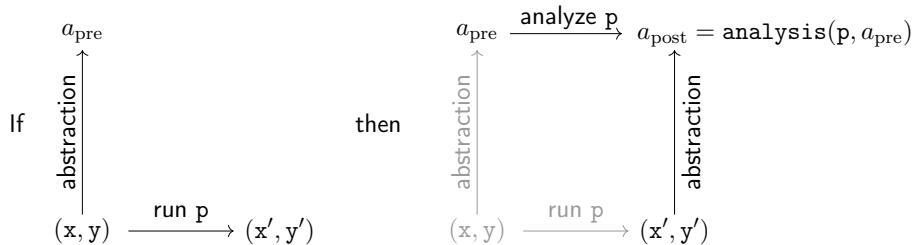


Figure: Sound analysis of a program p

Abstract Semantics Computation

Recall the example language

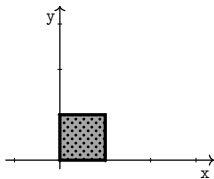
$p ::= \text{init}(\mathfrak{R})$	initialization, with a state in \mathfrak{R}
$\text{translation}(u, v)$	translation by vector (u, v)
$\text{rotation}(u, v, \theta)$	rotation defined by center (u, v) and angle θ
$p ; p$	sequence of operations
$\{p\} \text{or} \{p\}$	non-deterministic choice
$\text{iter}\{p\}$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

Approach

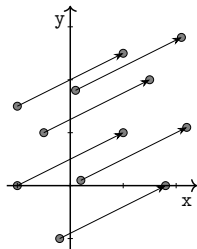
A sound analysis for a program is constructed by computing sound abstract semantics of the program's components.

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

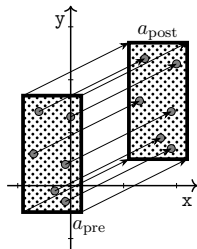
- Select, if any, the best abstraction of the region \mathfrak{R} .
- For the example program with the intervals or convex polyhedra abstract domains, analysis of $\text{init}([0, 1] \times [0, 1])$ is



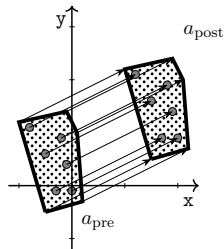
$\text{analysis}(\text{init}(\mathfrak{R}), a) = \text{best abstraction of the region } \mathfrak{R}$

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

(a) Concrete semantics

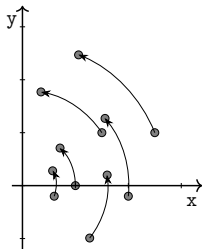


(b) Intervals

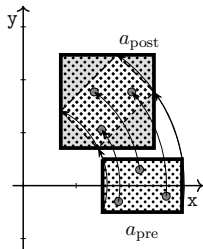


(c) Convex polyhedra

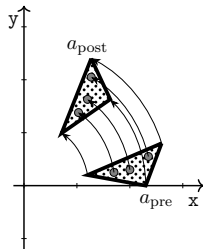
$\text{analysis}(\text{translation}(u, v), a) = \begin{cases} \text{return an abstract state that contains} \\ \text{the translation of } a \end{cases}$

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

(d) Concrete semantics

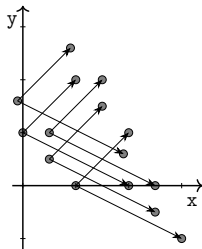


(e) Intervals

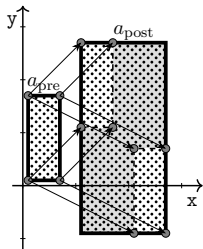


(f) Convex polyhedra

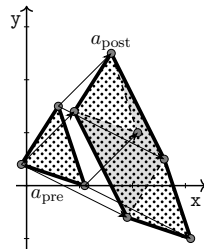
$$\text{analysis}(\text{rotation}(u, v, \theta), a) = \begin{cases} \text{return an abstract state that contains} \\ \text{the rotation of } a \end{cases}$$

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

(g) Concrete semantics



(h) Intervals



(i) Convex polyhedra

$$\text{analysis}(\{p_0\} \text{ or } \{p_1\}, a) = \text{union}(\text{analysis}(p_1, a), \text{analysis}(p_0, a))$$

Abstract Semantics Computation: $p_0 ; p_1$

$$\text{analysis}(p_0; p_1, a) = \text{analysis}(p_1, \text{analysis}(p_0, a))$$

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

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

$$\begin{aligned} & \{\} \\ & \text{or}\{p\} \\ & \text{or}\{p; p\} \\ & \text{or}\{p; p; p\} \\ & \text{or}\{p; p; p; p\} \\ & \vdots \end{aligned}$$

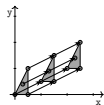
Abstract Semantics Computation: $\text{iter}\{p\}$ (2/5)

Example (Abstract iteration)

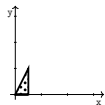
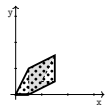
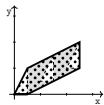
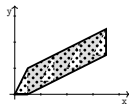
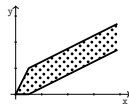
```

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: $\text{iter}\{p\}$ (3/5)

Recall

$$\begin{aligned}\text{iter}\{p\} &= \{\} \text{ or } \{p\} \text{ or } \{p;p\} \text{ or } \dots \\ &= \lim_i p_i\end{aligned}$$

where

$$p_0 = \{\} \quad p_{k+1} = p_k \text{ or } \{p_k;p\}$$

Hence,

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

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

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

Example (Abstract iteration with widening)

```

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

```

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

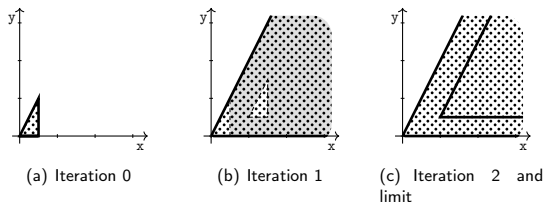


Figure: Abstract iteration with widening

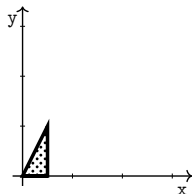
Abstract Semantics Computation: $\text{iter}\{p\}$ (5/5)

Example (Loop unrolling)

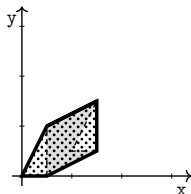
```

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

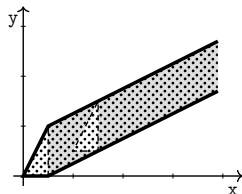
```



(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 `analysis(p, a)` is finitely computable and sound.

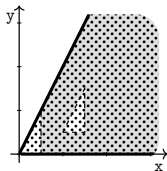
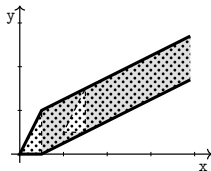
$$\begin{aligned}
 \text{analysis}(\text{init}(\mathfrak{R}), a) &= \text{best abstraction of the region } \mathfrak{R} \\
 \text{analysis}(\text{translation}(u, v), a) &= \begin{cases} \text{return an abstract state that contains} \\ \text{the translation of } a \end{cases} \\
 \text{analysis}(\text{rotation}(u, v, \theta), a) &= \begin{cases} \text{return an abstract state that contains} \\ \text{the rotation of } a \end{cases} \\
 \text{analysis}(\{p_0\} \text{ or } \{p_1\}, a) &= \text{union}(\text{analysis}(p_1, a), \text{analysis}(p_0, a)) \\
 \text{analysis}(p_0; p_1, a) &= \text{analysis}(p_1, \text{analysis}(p_0, a)) \\
 \text{analysis}(\text{iter}\{p\}, a) &= \begin{cases} R \leftarrow a; \\ \text{repeat} \\ \quad T \leftarrow R; \\ \quad R \leftarrow \text{widen}(R, \text{analysis}(p, R)); \\ \text{until inclusion}(R, T) \\ \text{return } T; \end{cases}
 \end{aligned}$$

Sound analysis

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

Verification of the Property of Interest

- Does program compute a point inside no-fly zone \mathcal{D} ?
- Need to collect the set of reachable points.
- Run `analysis(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 $\mathfrak{R} \cap \mathcal{D} = \emptyset$, then 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 .

$$\llbracket AB \rrbracket = \dots \llbracket A \rrbracket \dots \llbracket B \rrbracket \dots$$

- ▶ 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

$$\llbracket AB \rrbracket = \{s_0 \hookrightarrow s_1 \hookrightarrow \dots, \dots\}$$