

形式化方法导引

第 6 章 案例分析

6.2 Software Analysis: Abstract Interpretation | CEGAR

黄文超

<https://faculty.ustc.edu.cn/huangwenchao>

→ 教学课程 → 形式化方法导引

Software Analysis

1. Abstract Interpretation

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70 : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

回顾:

- $G (y \text{ in } (0..69))$
 - return false
- $G (y \text{ in } (0..71))$

Deductive verifiers require annotations (e.g., *loop invariants*) from users

- Fortunately, many techniques that can automatically learn *loop invariants*
- A common framework for this purpose is *Abstract Interpretation* (AI)

Software Analysis

1. Abstract Interpretation

```
MODULE main
VAR
  y : 0..15000;
ASSIGN
  init(y) := 0;
TRANS
  case
    y=70 : next(y)=0;
    TRUE : next(y)=y+1;
  esac
LTLSPEC
  G (y in (0..70))
```

回顾:

- $G (y \text{ in } (0..69))$
 - return false
- $G (y \text{ in } (0..71))$

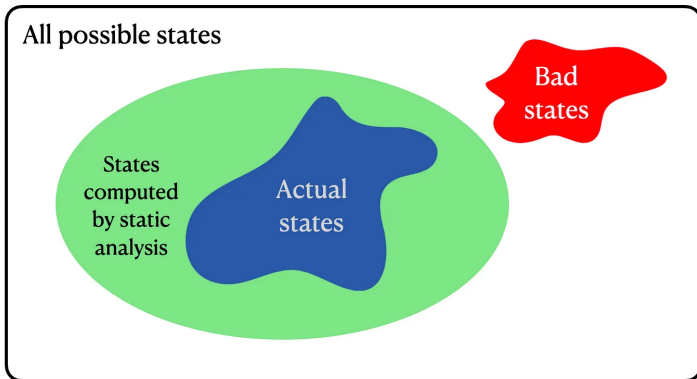
Deductive verifiers require annotations (e.g., *loop invariants*) from users

- Fortunately, many techniques that can automatically learn *loop invariants*
- A common framework for this purpose is *Abstract Interpretation* (AI)

Software Analysis

1. Abstract Interpretation

- *Abstract interpretation* forms the *basis* of *most static analyzers*
 - A framework for computing *over-approximations* of program states

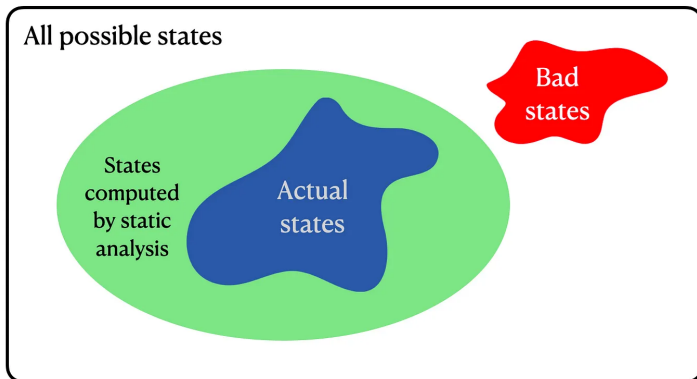


- Cons: *Cannot* reason about the *exact* program behavior
- Pros: It *can* be *enough* to prove program correctness

Software Analysis

1. Abstract Interpretation

- *Abstract interpretation* forms the *basis* of *most static analyzers*
 - A framework for computing *over-approximations* of program states



- Cons: *Cannot* reason about the *exact* program behavior
- Pros: It *can* be *enough* to prove program correctness

Software Analysis

1. Abstract Interpretation

- Motivation Example: Insertion Sort

```
1 for i=1 to 99 do
2
3     p := T[i]; j := i+1;
4
5     while j <= 100 and T[j] < p do
6
7         T[j-1] := T[j]; j := j+1;
8
9     end;
10
11     T[j-1] := p;
12 end;
```

问: Is there any out of bound array access?

Software Analysis

1. Abstract Interpretation

- Motivation Example: Insertion Sort

```
1 for i=1 to 99 do
2   // i ∈ [1, 99]
3   p := T[i]; j := i+1;
4   // i ∈ [1, 99], j ∈ [2, 100]
5   while j <= 100 and T[j] < p do
6     // i ∈ [1, 99], j ∈ [2, 100]
7     T[j-1] := T[j]; j := j+1;
8     // i ∈ [1, 99], j ∈ [3, 101]
9   end;
10  // i ∈ [1, 99], j ∈ [2, 101]
11  T[j-1] := p;
12 end;
```

问: Is there any out of bound array access?

答: No

- by [interval analysis](#), using an AI tool, e.g., [Apron](#)

The AI Recipe

- ① Define **abstract domain** - fixes “shape” of the invariants
 - e.g., $c_1 \leq x \leq c_2$ (*intervals*), or $\pm x \pm y \leq c$ (octagons)
- ② Define **abstract semantics (transformers)**
 - Define how to symbolically execute each statement in the chosen abstract domain
 - Must be sound wrt to concrete semantics
- ③ Iterate abstract transformers until **fixed point**
 - The fixed-point is an over-approximation of program behavior

Software Analysis

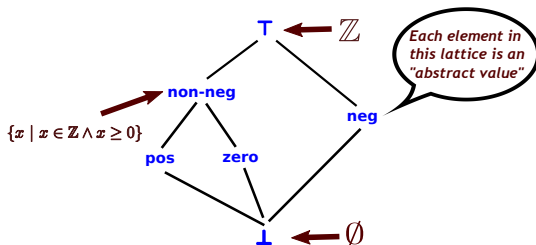
1. Abstract Interpretation | 1.1 Abstract Domain

Simple Example: Sign Domain

Suppose we want to infer invariants of the form $x \bowtie 0$ where

- $\bowtie \in \{\geq, =, >, <\}$
- i.e., zero, non-negative, positive, negative

This corresponds to the following abstract domain represented as *lattice*:



Lattice is a *partially ordered set*

- i.e., (S, \sqsubseteq) , where
- each pair of elements has
 - a least upper bound
 - i.e., **join** \sqcup
 - a greatest lower bound
 - i.e., **meet** \sqcap

Software Analysis

1. Abstract Interpretation | 1.1 Abstract Domain

The “meaning” of abstract domain is given by *abstraction* and *concretization* functions that relate concrete and abstract values

Concretization function (γ)

It maps each *abstract value* to *sets of concrete elements*

- $\gamma(\mathbf{pos}) = \{x \mid x \in \mathbb{Z} \wedge x > 0\}$

Abstraction function (α)

It maps *sets of concrete elements* to the *most precise value* in the abstract domain

- $\alpha(\{2, 10, 0\}) = \mathbf{non-neg}$
- $\alpha(\{3, 99\}) = \mathbf{pos}$
- $\alpha(\{-3, 2\}) = \top$

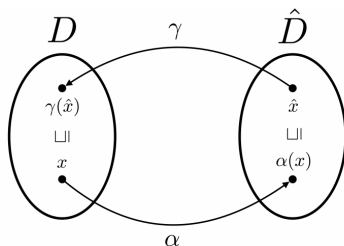
Software Analysis

1. Abstract Interpretation | 1.1 Abstract Domain

Important requirement: concrete domain D and abstract domain \hat{D} must be related through *Galois connection*:

Galois connection

$$\forall x \in D, \forall \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq \hat{x} \Leftrightarrow x \sqsubseteq \gamma(\hat{x})$$



Intuitively, this says that α, γ respect the orderings of D, \hat{D}

Software Analysis

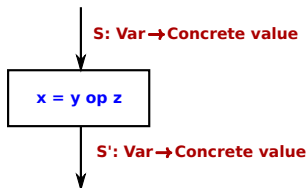
1. Abstract Interpretation | 1.2 Abstract Semantics

Step 2: Abstract Semantics

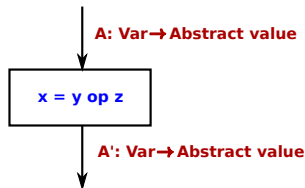
Define *abstract* transformers (i.e., *semantics*) for each statement, given abstract domain, α , γ

- Describes how statements affect our abstraction
- Abstract counter-part of *operational semantics* rules

Operational Semantics



Abstract Semantics



Software Analysis

1. Abstract Interpretation | 1.2 Abstract Semantics

- Back to Our Example, we can define abstract transformer for $x = y + z$ as follows:

	pos	neg	zero	non-neg	\top	\perp
pos	pos	\top	pos	pos	\top	\perp
neg	\top	neg	neg	\top	\top	\perp
zero	pos	neg	zero	non-neg	\top	\perp
non-neg	pos	\top	non-neg	non-neg	\top	\perp
\top	\top	\top	\top	\top	\top	\perp
\perp	\perp	\perp	\perp	\perp	\perp	\perp

Soundness of Abstract Transformers

- *Important requirement*: Abstract semantics must be *sound* wrt (i.e., faithfully models) the concrete semantics

Soundness of \hat{F}

If F is the concrete transformer and \hat{F} is its abstract counterpart, soundness of \hat{F} means:

$$\forall x \in D, \forall \hat{x} \in \hat{D}. \alpha(x) \sqsubseteq \hat{x} \Rightarrow \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})$$

Note: recall Galois connection

- In other words, If \hat{x} is an overapproximation of x , then $\hat{F}(\hat{x})$ is an over-approximation of $F(x)$

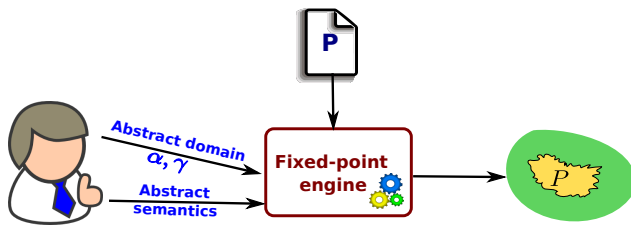
Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point

Step 3: Fixed-Point Computation

Repeated symbolic execution of the program using abstract semantics until our approximation of the program reaches an *equilibrium*:

$$\bigsqcup_{i \in \mathbb{Z}} \hat{F}^i(\perp)$$



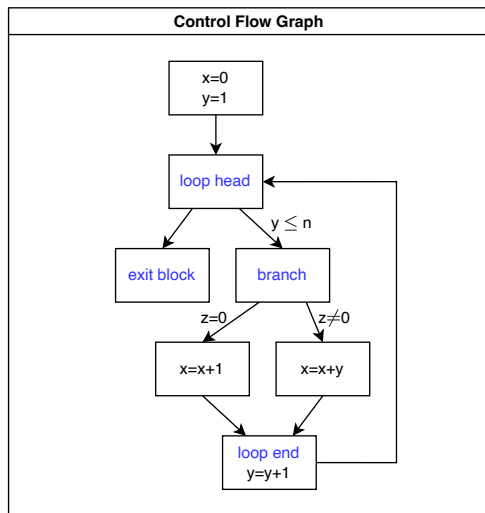
Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point

An example:

```
1 x = 0;  
2 y = 1;  
3 while (y <= n) {  
4     if (z == 0) {  
5         x = x+1;  
6     }  
7     else {  
8         x = x + y;  
9     }  
10    y = y+1;  
11 }
```

- Specification: Is x *always non-negative* inside the loop?



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point

Least fixed-point

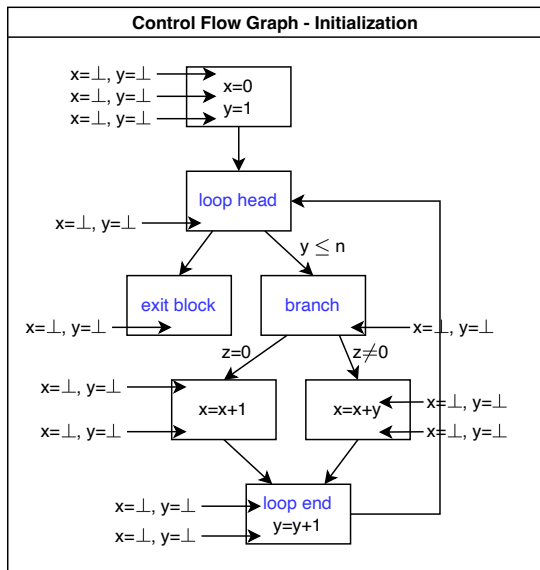
- *Start with underapproximation*
- Want to compute abstract values at every program point
- Grow the approximation until it stops growing

Least fixed-point

- ① *Initialize* all abstract states to \perp
 - ② *Repeat* until no abstract state changes at any program point:
 - Compute abstract state on entry to *a basic block B* by taking the *join* of B's *predecessors*
 - *Symbolically execute* each basic block using abstract semantics
- 性质: *Assuming correctness* of your abstract semantics, the *least fixed point* is an *overapproximation* of the program!

Software Analysis

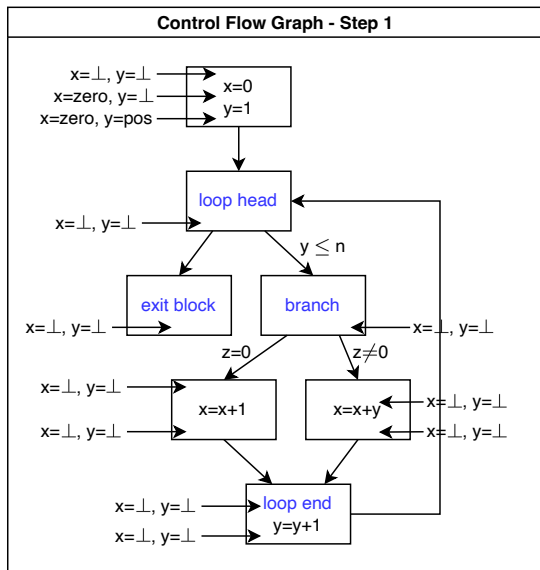
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

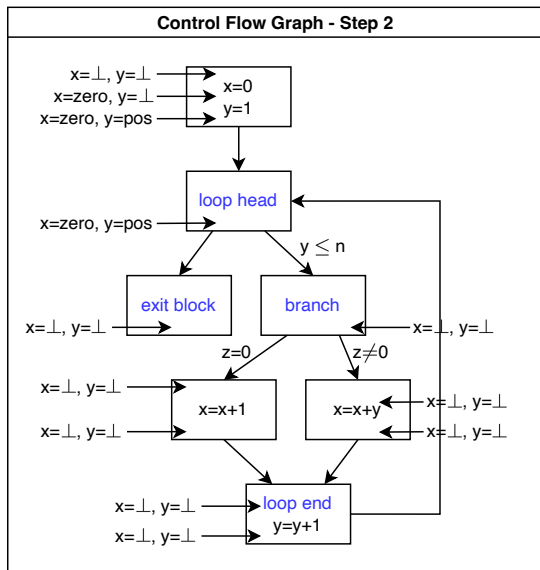
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

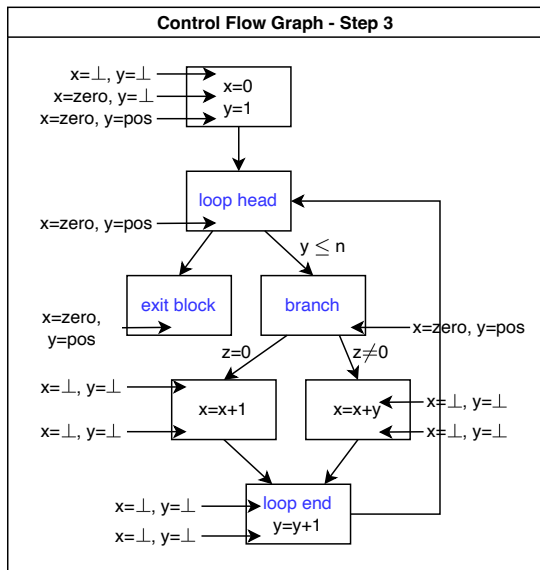
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

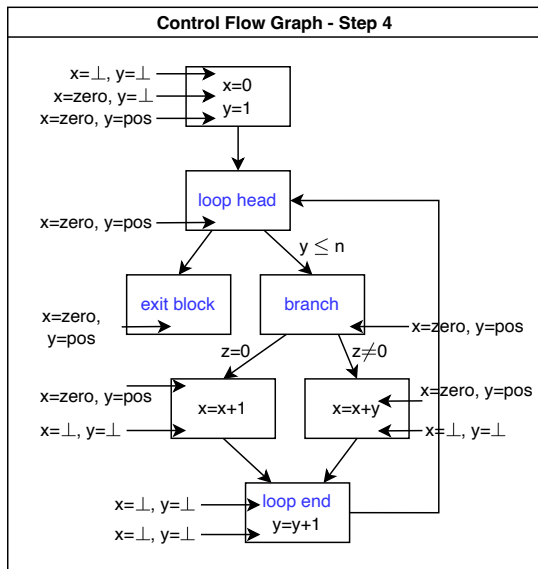
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

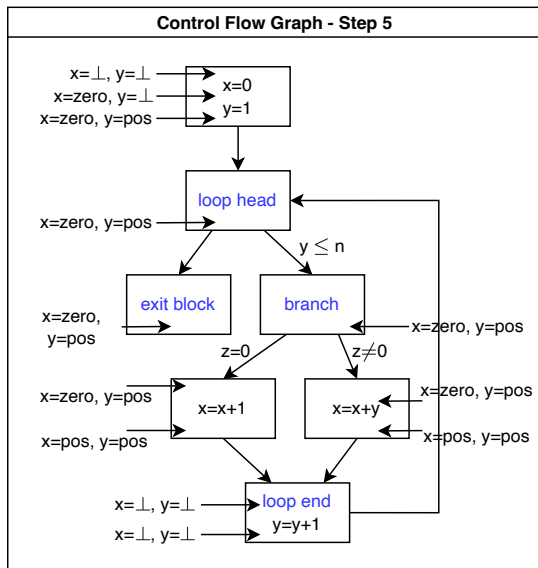
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

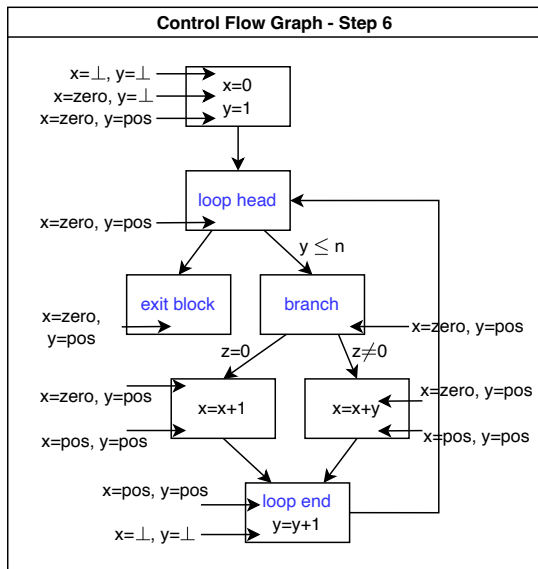
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

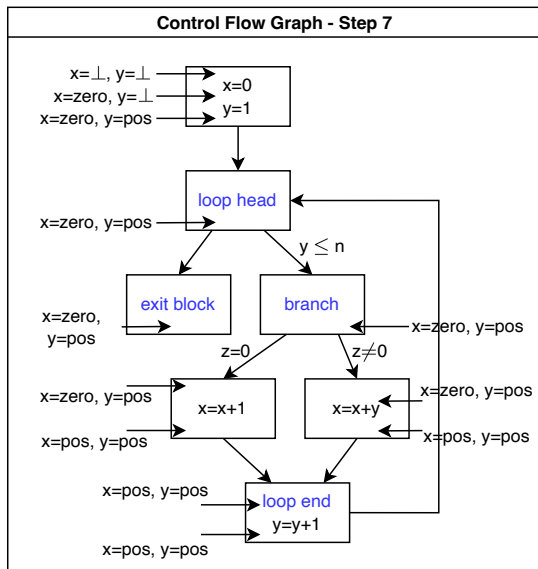
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

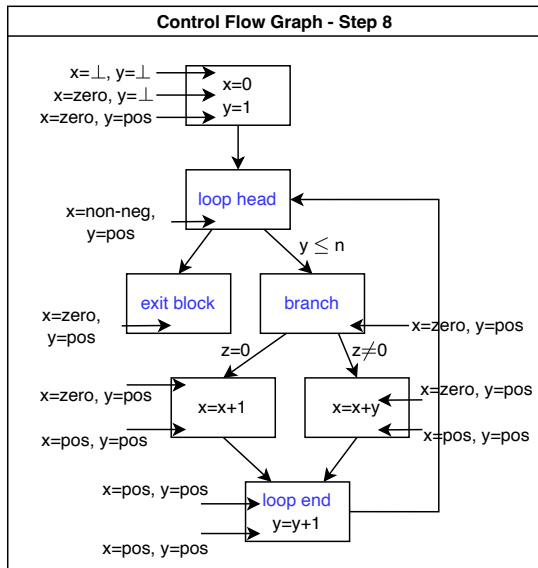
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

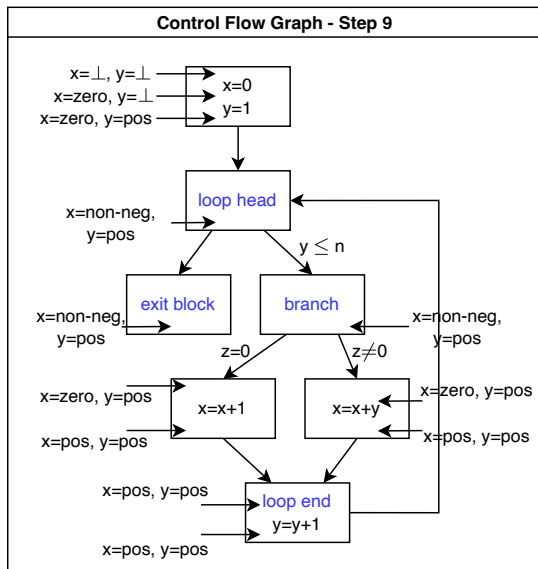
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

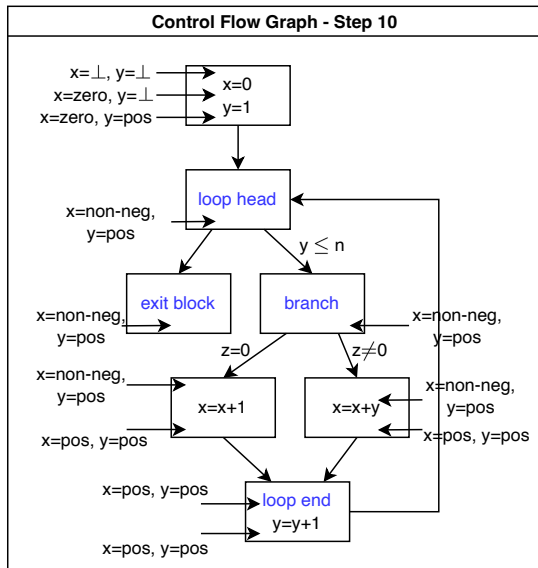
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

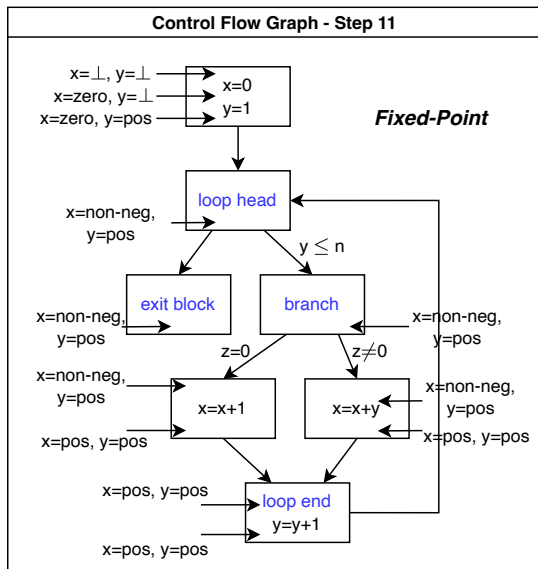
1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Least fixed-point



Least fixed-point

问: does this computation *always terminate*?

- Yes if the lattice has *finite height*;
- otherwise, it *might not*

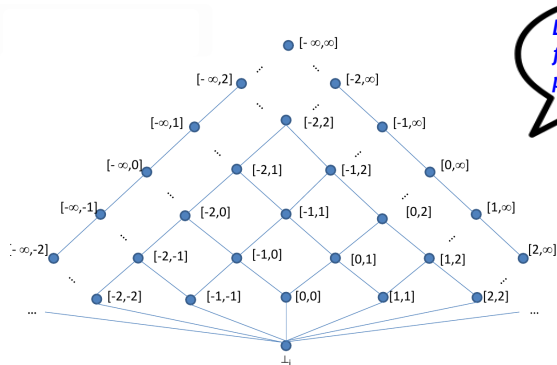
Solution: *widening operators*

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

Interval Analysis

- In the interval domain, abstract values are of the form $[c_1, c_2]$ where c_1 is a lower bound and c_2 has an upper bound
- If the abstract value for x is $[1, 3]$ at some program point P , this means $1 \leq x \leq 3$ is an *invariant* of P



*Does not have
finite-height
property!*

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

Requirements on **widening** ∇ operator

- 1 $\forall a, b \in \hat{D}. a \sqcup b \sqsubseteq a \nabla b$
- 2 For all increasing chains $d_0 \sqsubseteq d_1 \sqsubseteq \dots$, the ascending chains $d_0^\nabla \sqsubseteq d_1^\nabla \sqsubseteq \dots$ eventually *stabilizes* where $d_0^\nabla = d_0$ and

$$d_{i+1}^\nabla = d_i^\nabla \nabla d_{i+1}$$

性质

- Overapproximate least-fixed-point by using widening operator rather than *join*
 - Sound and guaranteed to *terminate*
- This is called *post-fixed-point*

例: Widening in Interval Domain

$$[a, b] \nabla \perp = [a, b]$$

$$\perp \nabla [a, b] = [a, b]$$

$$[a, b] \nabla [c, d] = [(c < a? -\infty : a), (b < d? +\infty : b)]$$

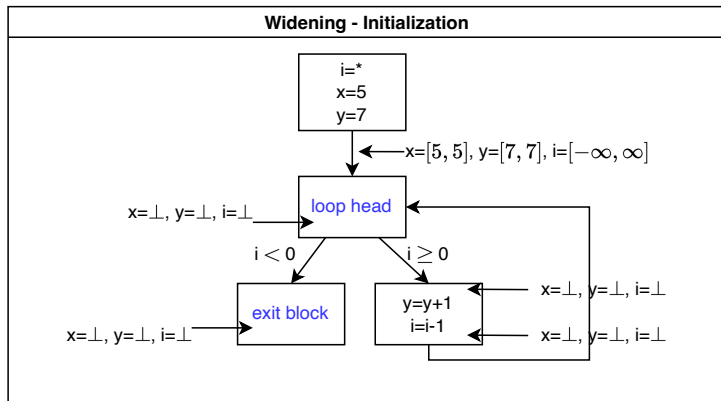
作业 1

- $[1, 2] \nabla [0, 2] =$
- $[0, 2] \nabla [1, 2] =$
- $[1, 5] \nabla [1, 5] =$
- $[2, 3] \nabla [2, 4] =$

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

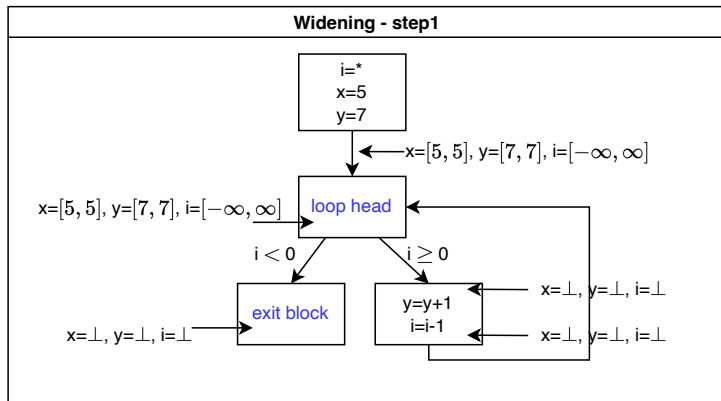
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

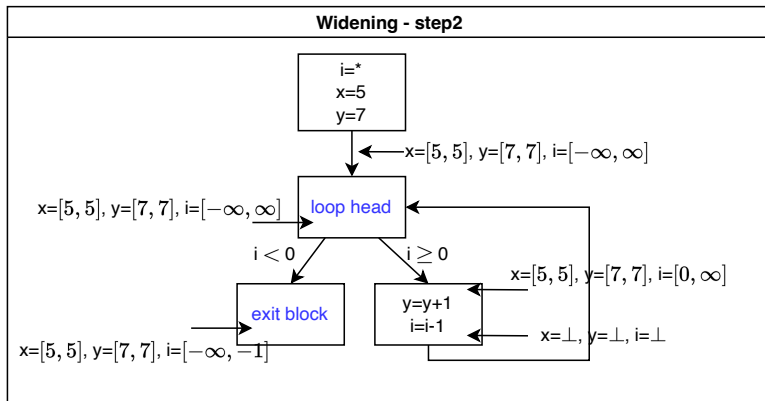
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

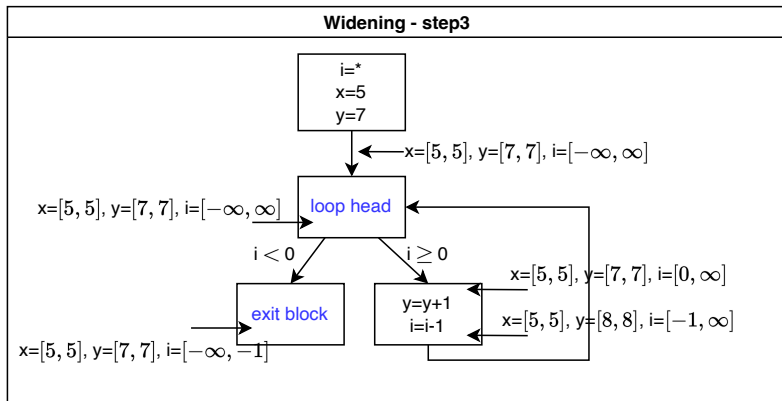
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

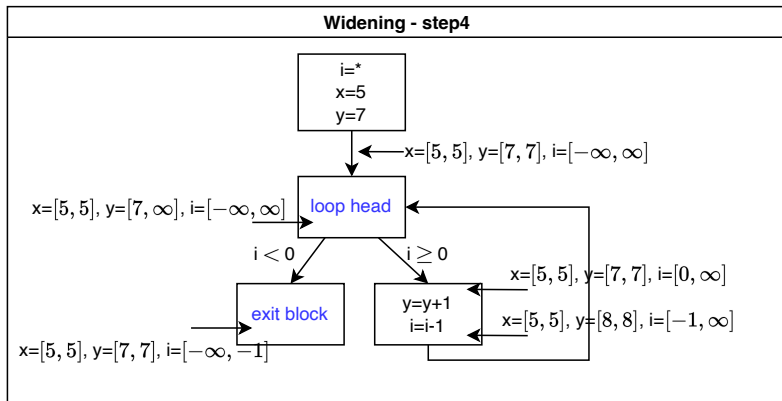
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

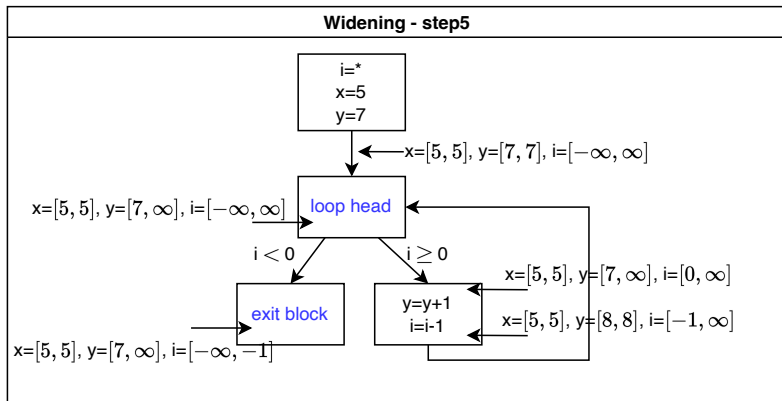
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

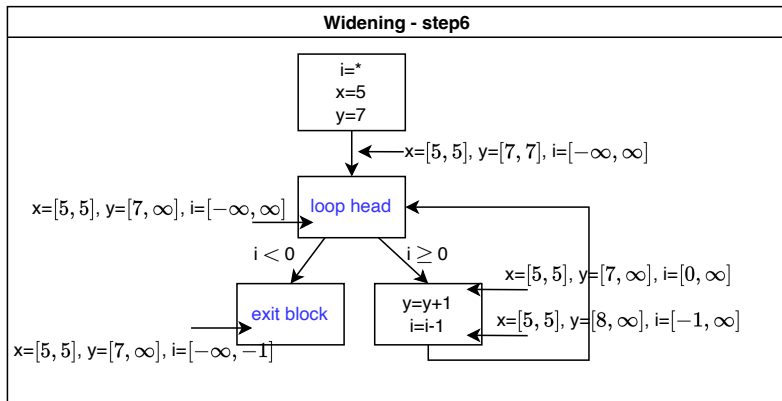
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

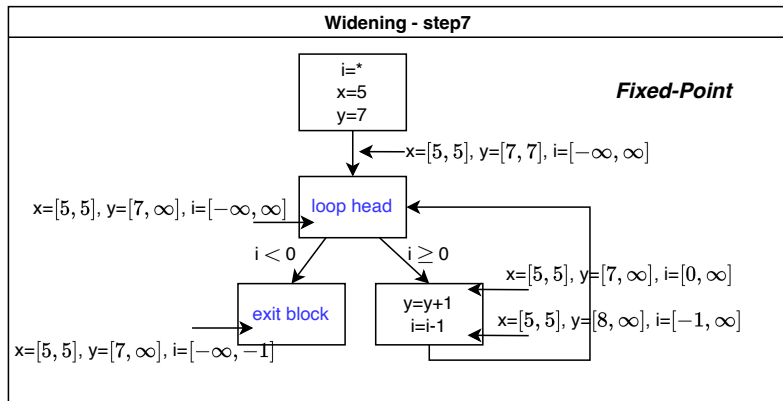
Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Widening

Example with widening



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

- 问题: In many cases, *widening overshoots* and generates imprecise results
- 例:

```
1 x=1;  
2 while (*) {  
3     x = 2;  
4 }
```

- After widening, x 's abstract value will be $[1, \infty]$ after the loop; but more precise value is $[1, 2]$
- 解决方法: After finding a post-fixed-point (using widening), have a second pass using a *narrowing* operator

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

- 问题: In many cases, *widening overshoots* and generates imprecise results
- 例:

```
1 x=1;  
2 while (*) {  
3     x = 2;  
4 }
```

- After widening, x 's abstract value will be $[1, \infty]$ after the loop; but more precise value is $[1, 2]$
- 解决方法: After finding a post-fixed-point (using widening), have a second pass using a *narrowing* operator

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

Requirements on **Narrowing** \triangle operator (Recall Widening ∇)

- ① $\forall x, y \in \hat{D}. (y \sqsubseteq x) \Rightarrow y \sqsubseteq (x \triangle y) \sqsubseteq x$
- ② For all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \dots$, the chains y_0, y_1, \dots eventually *stabilizes* where $y_0 = x_0$ and

$$y_{i+1} = y_i \triangle x_{i+1}$$

例: **Narrowing** in **Interval Domain**

$$[a, b] \triangle \perp = [a, b]$$

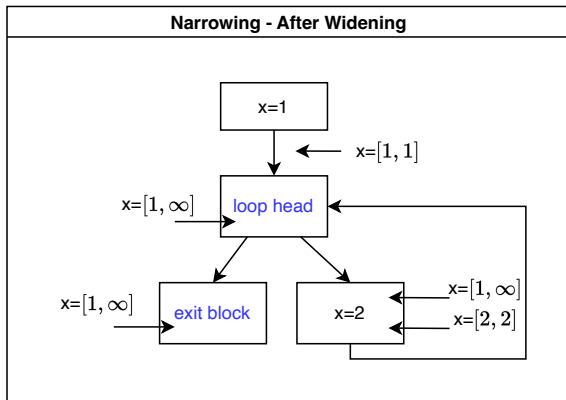
$$\perp \triangle [a, b] = [a, b]$$

$$[a, b] \triangle [c, d] = [(a = -\infty ? c : a), (b = \infty ? d : b)]$$

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

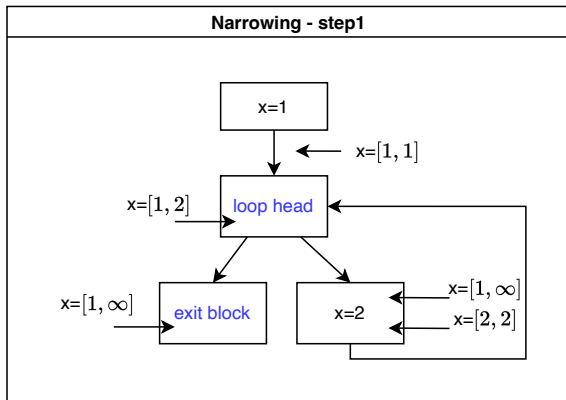
Example with narrowing



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

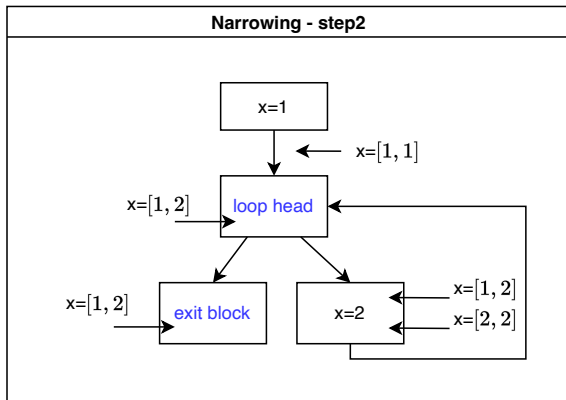
Example with narrowing



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

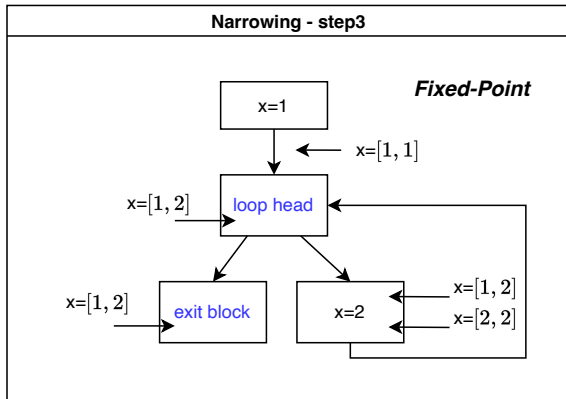
Example with narrowing



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Narrowing

Example with narrowing



Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Relational Abstract Domains

- Both the sign and interval domain are *non-relational domains*
 - i.e., do not relate different program variables
- *Relational domains* track relationships between variables
 - more powerful
- A motivating example

```
1   x=0; y=0;
2   while (*) {
3       x = x+1; y = y+1;
4   }
5   assert(x=y);
```

- *Cannot* prove this assertion using *interval domain*

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Relational Abstract Domains

- *Karr's domain*: Tracks equalities between variables (e.g., $x = 2y + z$)
- *Octagon domain*: Constraints of the form $\pm x \pm y \leq c$
- *Polyhedra domain*: Constraints of the form $c_1x_1 + \dots c_nx_n \leq c$
- Polyhedra domain most precise among these, but can be expensive (exponential complexity)
- Octagons less precise but cubic time complexity

回顾 & 作业 2

Software Analysis

1. Abstract Interpretation | 1.3 Fixed Point | Relational Abstract Domains

- *Karr's domain*: Tracks equalities between variables (e.g., $x = 2y + z$)
- *Octagon domain*: Constraints of the form $\pm x \pm y \leq c$
- *Polyhedra domain*: Constraints of the form $c_1x_1 + \dots c_nx_n \leq c$
- Polyhedra domain most precise among these, but can be expensive (exponential complexity)
- Octagons less precise but cubic time complexity

回顾 & 作业 2

A method of software model checking:

- Counterexample-Guided Abstraction Refinement (*CEGAR*)

Keywords:

- Predicate Abstraction
 - Predicate Abstraction Lattice
- Abstract Transformers
 - Strongest Postcondition
- Refinement

Example

```
1 x:=0; y:= 0;  
2 while(x<100)  
3 {  
4     x := x+1;  
5     y := y+1;  
6 }  
7 assert(y = 100);
```

- Predicate set $\mathcal{P} = \{x < 100, y = 100\}$

Predicate Abstraction

Given a set of predicates $\mathcal{P} = \{p_1, \dots, p_n\}$, predicate abstraction computes for *every program location*, an abstract value $[b_1, \dots, b_n]$ where:

- b_i indicates whether p_i holds or not at that location
- values of b_i drawn from the set $\{0, 1, *\}$ where $*$ indicates unknown
- In the example, at Line 1, $[b_1, b_2] = [1, 0]$
- In other words, we have an abstract domain where each element is a formula $\bigwedge_i l_i$ (sometimes called a *cube*), where $l_i = p_i \mid \neg p_i$

性质: Predicate Abstraction Lattice

Given predicates \mathcal{P} , $(Cubes(\mathcal{P}), \Rightarrow)$ forms a complete lattice

- $Cubes(\mathcal{P})$ is any formula $\bigwedge_i p_i$ where p_i is a predicate or the negation of a predicate in \mathcal{P}
- In other words, we have $\phi_1 \sqsubseteq \phi_2$ iff $\phi_1 \Rightarrow \phi_2$
 - e.g., $p_1 \wedge p_2 \sqsubseteq p_1$
- 练习: How do we compute $\phi_1 \sqcup \phi_2$?,
 - $(p_1 \wedge p_2) \sqcup p_1$?
 - $(p_1 \wedge p_2) \sqcup \neg p_1$

Abstract Transformers

Given a statement S and cube ϕ , define abstract transformer $post^\#(S, \phi)$ to be the strongest cube ϕ' over \mathcal{P} such that:

$$sp(S, \phi) \Rightarrow \phi'$$

where sp is the *strongest post-condition* of S wrt to ϕ

Strongest Postcondition $sp(S, \phi)$

Executing statement S on any state s_0 in the ϕ region must result in a state s in the $sp(S, \phi)$ region

- $sp(\text{assume } c, \phi) \Leftrightarrow c \wedge \phi$
- $sp(v := e[v], \phi[v]) \Leftrightarrow \exists v_0. v = e[v_0] \wedge \phi[v_0]$

Example: Given $\mathcal{P} = \{x = y, x \neq y, x \geq y\}$, compute $post^\#(x := x + 1, x = y)$?

The answer is $[b_1, b_2, b_3] = [0, 1, 1]$

- $sp(x := x + 1, x = y) \Leftrightarrow (\exists x_0. x = x_0 + 1 \wedge x_0 = y) \Leftrightarrow (x = y + 1)$
- $\phi' \equiv (b_1 = 0 \wedge b_2 = 1 \wedge b_3 = 1)$
 - since $(x = y + 1) \Rightarrow \phi'$

作业 3: Practice in program, compute

- $post^\#(x := x + 1, x < 100)$
- $post^\#(x := x + 1, x < 100 \wedge y = 100)$

Motivation for CEGAR

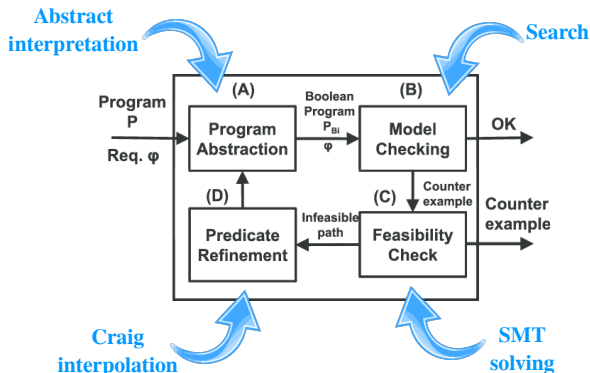
- Predicate abstraction is very *sensitive* to the set of predicates
 - If you choose the right set, verification succeeds; otherwise, it fails
- The *CEGAR* paradigm allows *automatically* and *iteratively discovering* the *right set* of predicates

Software Analysis

2. CEGAR

Key Steps:

- Program Abstraction
- Model Checking
- Feasibility Check
- Refinement



Software Analysis

2. CEGAR | Program Abstraction

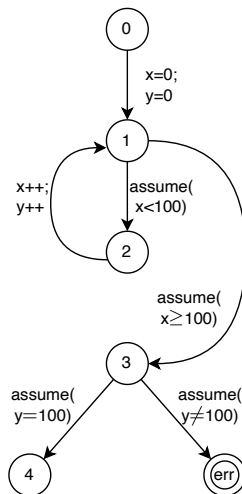
Motivation for *Program Abstraction*

- Given a program P , the state is a tuple l, v_1, v_2, \dots, v_n , where
 - l is the **control location**
 - v_i denotes the value of i th variable
- The **state space** is large or even infinite!

Idea: construct a so-called **boolean program** via *predicate abstraction*

- Replace *concrete states* with *predicates*
- Operate over control-flow automaton (CFA)
 - Like CFG but nodes/edges are flipped + explicit error locations

CFA (Original)



Software Analysis

2. CEGAR | Program Abstraction

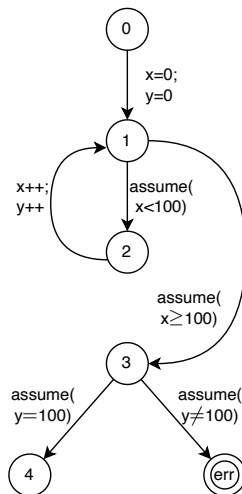
Motivation for *Program Abstraction*

- Given a program P , the state is a tuple l, v_1, v_2, \dots, v_n , where
 - l is the **control location**
 - v_i denotes the value of i th variable
- The **state space** is large or even infinite!

Idea: construct a so-called **boolean program** via **predicate abstraction**

- Replace **concrete states** with **predicates**
- Operate over control-flow automaton (CFA)
 - Like CFG but nodes/edges are flipped + explicit error locations

CFA (Original)



Software Analysis

2. CEGAR | Program Abstraction

Program Abstraction

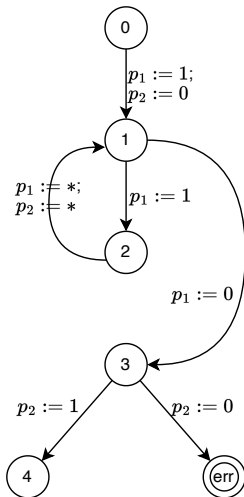
- state space
 - From $6 * 2^{32} * 2^{32}$ to $6 * 2 * 2$

问: How to translate the program into a boolean program?

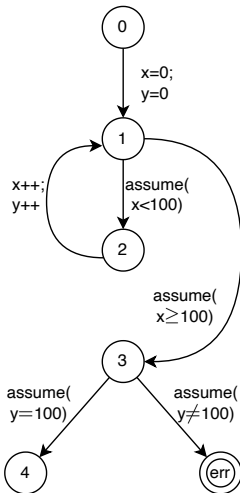
作业 4: Translate statements in CFA

- $1 \rightarrow 2$
- $2 \rightarrow 1$

CFA (Predicate Abstraction)



CFA (Original)



Translating Statements

Given statement S and boolean b representing predicate p ,

- Compute the weakest cubes P_1, P_2 over P such that
 - $P_1 \Rightarrow \text{wp}(S, p)$ and $P_2 \Rightarrow \text{wp}(S, \neg p)$
- Translate the statement S

```
1 if (P1) b := true
2 else if (P2) b := false
3 else b := *
```

Weakest Precondition (回顾: Strongest Postcondition)

Every state s on which executing statement S leads to a state s' in the ϕ region must be in the $\text{wp}(S, \phi)$ region

- $\text{wp}(\text{assume } c, \phi) \Leftrightarrow c \rightarrow \phi$
- $\text{wp}(v := e, \phi[v]) \Leftrightarrow \phi[e]$

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```


Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

Software Analysis

2. CEGAR | Program Abstraction

- Example: Consider the predicates $\{x > 5, x < 5, y = 5\}$, how to translate the statement $x := y$ in the boolean program with variables b_1, b_2, b_3 ?
- $\text{wp}(x := y, x > 5) \Leftrightarrow (y > 5)$
- $\text{wp}(x := y, x < 5) \Leftrightarrow (y < 5)$
- $\text{wp}(x := y, y = 5) \Leftrightarrow (y = 5)$
- For b_1 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_1 := 0$, else $b_1 := *$
- For b_2 , since $b_3 \Rightarrow \neg(y > 5)$, translation: if (b_3) $b_2 := 0$, else $b_2 := *$
- For b_3 , since $b_3 \Rightarrow (y = 5)$ and $\neg b_3 \Rightarrow \neg(y = 5)$, translation: (Empty)

Totally, the translated statements of $x := y$ are

```
1  if (b3) b1:=0, b2:=0
2  else b1:=*, b2:=*
```

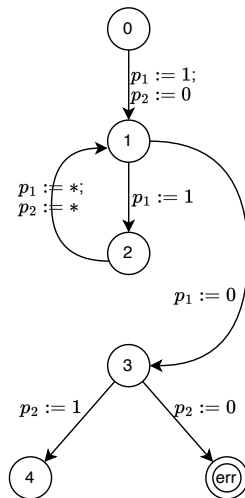
Model Checking

- *Initial states:*

$(0, p_1, p_2), (0, \neg p_1, p_2), (0, p_1, \neg p_2), (0, \neg p_1, \neg p_2)$

- There is a *transition* from (l, b_1, \dots, b_n) to (l', b'_1, \dots, b'_n) *iff*:
 - There must be a transition from l to l' labeled with S
 - The formula $\text{sp}(S, \bigwedge_i b_i) \wedge \bigwedge_i b'_i$ must be satisfiable. (query SAT solver)

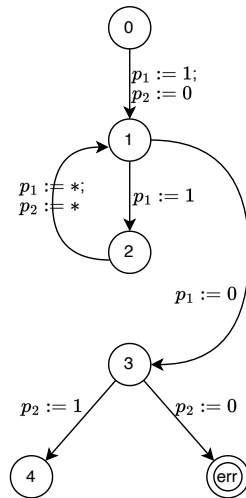
CFA (Predicate Abstraction)



练习: Which of these transition exist in the state transition graph?

- $(1, p_1, p_2)$ to $(3, \neg p_1, p_2)$
- $(1, p_1, p_2)$ to $(3, p_1, p_2)$
- $(3, \neg p_1, p_2)$ to $(err, \neg p_1, \neg p_2)$

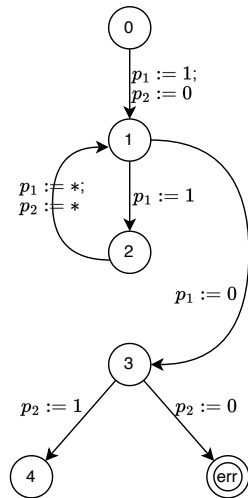
CFA (Predicate Abstraction)



练习: Which of these transition exist in the state transition graph?

- $(1, p_1, p_2)$ to $(3, \neg p_1, p_2)$
 - Yes
- $(1, p_1, p_2)$ to $(3, p_1, p_2)$
- $(3, \neg p_1, p_2)$ to $(\text{err}, \neg p_1, \neg p_2)$

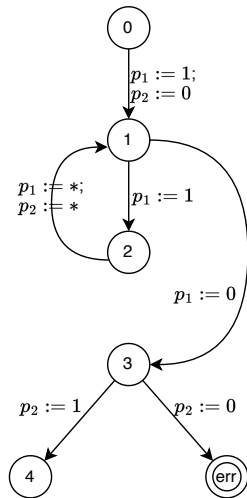
CFA (Predicate Abstraction)



练习: Which of these transition exist in the state transition graph?

- $(1, p_1, p_2)$ to $(3, \neg p_1, p_2)$
 - Yes
- $(1, p_1, p_2)$ to $(3, p_1, p_2)$
 - No
- $(3, \neg p_1, p_2)$ to $(\text{err}, \neg p_1, \neg p_2)$

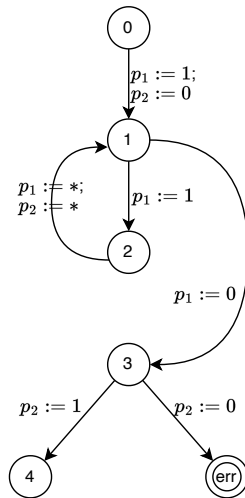
CFA (Predicate Abstraction)



练习: Which of these transition exist in the state transition graph?

- $(1, p_1, p_2)$ to $(3, \neg p_1, p_2)$
 - Yes
- $(1, p_1, p_2)$ to $(3, p_1, p_2)$
 - No
- $(3, \neg p_1, p_2)$ to $(\text{err}, \neg p_1, \neg p_2)$
 - Yes

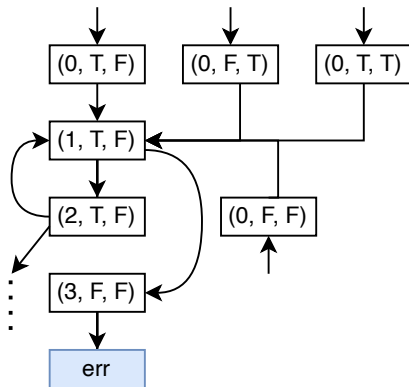
CFA (Predicate Abstraction)



Software Analysis

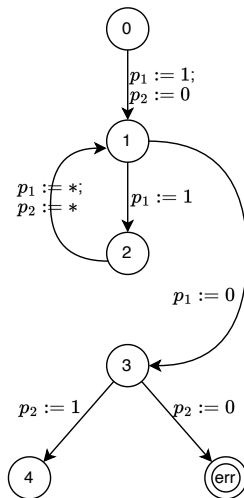
2. CEGAR | Model Checking

Partial transition system:



Verification outputs FALSE because *error state* is *reachable*!

CFA (Predicate Abstraction)



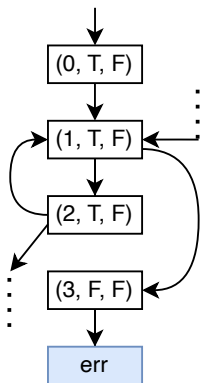
Feasibility Check

- *But* if the error state is reachable, this could be due to *imprecision* in the abstraction
 - i.e., current set of predicates may *not* be *fine-grain enough*
- To decide how to proceed, we need to check if the property is *actually* violated
- Fortunately, the model checker can provide a *counterexample* in the form of a *program trace*!

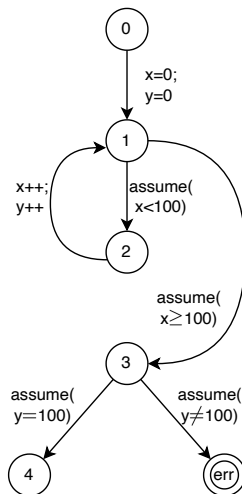
Software Analysis

2. CEGAR | Model Checking | Feasibility Check

Transition System



CFA (Original)



Counterexample Trace

- 1 `x := 0; y:=0;`
- 2 `assume(x >= 100);`
- 3 `assume(y != 100);`

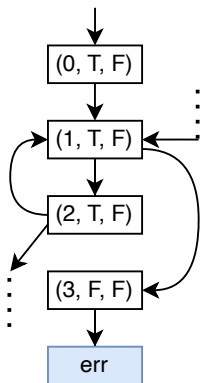
Clearly spurious
because the trace
formula is *UNSAT*:

$$\begin{aligned} &x = 0 \wedge \\ &y = 0 \wedge \\ &x \geq 100 \wedge \\ &y \neq 100 \end{aligned}$$

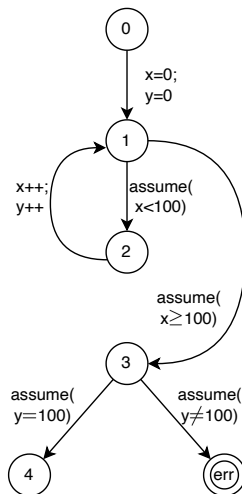
Software Analysis

2. CEGAR | Model Checking | Feasibility Check

Transition System



CFA (Original)



Counterexample Trace

```
1 x := 0; y:=0;  
2 assume(x>=100);  
3 assume(y!=100);
```

Clearly spurious
because the trace
formula is **UNSAT**:

$$\begin{aligned}x &= 0 \wedge \\y &= 0 \wedge \\x &\geq 100 \wedge \\y &\neq 100\end{aligned}$$

Refinement

- Goal: *prevent* the model checker from giving the *same* counterexample trace as before
- 问: How do we find predicates that will rule out this spurious trace?
- *Most basic idea*: Compute strongest postcondition for each statement in the counterexample trace; add these to set of predicates!

Eliminating counterexamples using SP

- Let $l_0 \rightarrow^{s_1} l_1 \rightarrow^{s_2} \dots \rightarrow^{s_n} l_n$ be a spurious counterexample trace
- Let p_0 be true, and define p_i as $\text{sp}(s_i, p_{i-1})$
- **Claim:** Adding p_1, \dots, p_n to \mathcal{P} will rule out this counterexample!
- **Why?** Consider any *potential* path in the transition system:

$$(l_0, \phi_0) \rightarrow^{s_1} (l_1, \phi_1) \rightarrow^{s_2} \dots \rightarrow^{s_n} (l_n, \phi_n)$$

- ① $\phi_i \Rightarrow p_i$ (note: See the proof in notes)
- ② It implies that such a path cannot exist in the transition system.
 - Why?

The *problem* of the most basic idea

- *Only* removes this counterexample trace
- Ideally, we want to learn predicates that allow us to remove *multiple* spurious traces
- *Trick*: We can learn more general predicates using a technique called *Craig interpolation*

Craig interpolation: Given an *unsatisfiable* formula $\phi_1 \wedge \phi_2$, a Craig interpolant is a formula ψ such that:

- $\phi_1 \Rightarrow \psi$
- $\text{UNSAT}(\phi_2 \wedge \psi)$
- ψ is over the common variables of ϕ_1 and ϕ_2

Interpolant Examples

$$\phi_1 \equiv x \leq w \wedge y \geq w \wedge z = x$$

$$\phi_2 \equiv y < t \wedge t = z$$

- Which of the following formulas are interpolants for $\phi_1 \wedge \phi_2$?

① $y \geq z$

② $y \geq x \wedge z = x$

③ $y > z$

How to learn by Craig interpolation?

- Let $l_0 \xrightarrow{s_1} l_1 \xrightarrow{s_2} \dots \xrightarrow{s_n} l_n$ be a spurious counterexample trace
- For simplicity, suppose the trace is in SSA form and suppose $\text{enc}(s_i)$ gives logical encoding of s_i 's semantics
- Then, we know that the following formula is *UNSAT*:

$$\text{enc}(s_1) \wedge \text{enc}(s_2) \wedge \dots \wedge \text{enc}(s_n)$$

- Now let ϕ_i^- denote the trace formula *up to* statement i and ϕ_i^+ denote the formula after i
- Then, for each location l_i , we have $\text{UNSAT } (\phi_i^- \wedge \phi_i^+)$ and *the interpolant gives predicates that are useful to track at l_i !*

Software Analysis

2. CEGAR | Model Checking | Refinement | Craig interpolation

- Consider the following counterexample trace that corresponds to executing loop body *once*:

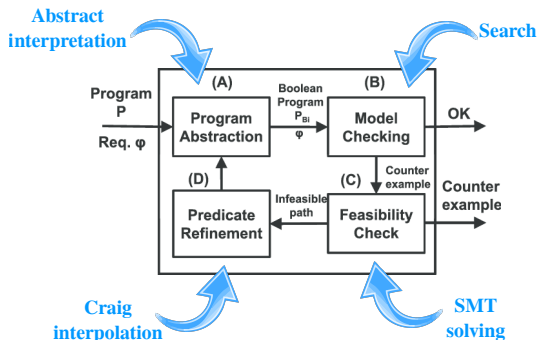
$$\left. \begin{array}{l} x_0 := 0; y_0 := 0; \\ \text{assume}(x_0 < 100); \\ x_1 := x_0 + 1; \\ y_1 := y_0 + 1; \\ \text{assume}(x_1 \geq 100); \\ \text{assume}(y_1 \neq 100); \end{array} \right\} \begin{array}{l} \boxed{x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 100} \\ \boxed{x_1 = x_0 + 1 \wedge y_1 = y_0 + 1} \end{array} \phi_1$$
$$\left. \begin{array}{l} \text{assume}(x_1 \geq 100); \\ \text{assume}(y_1 \neq 100); \end{array} \right\} \boxed{x_1 \geq 100 \wedge y_1 \neq 100} \phi_2$$

- Interpolant: $x_1 = y_1 \wedge x_1 \leq 100$
- Using the predicates in the interpolant, we *can now verify* the correctness of this program!

Per-location Abstraction

- In the *basic form* of predicate abstraction, we have a global set of predicates that we "track" *everywhere*
 - But not all predicates are useful everywhere...
- **Observation:** The interpolant tells us *which predicates are useful where!*
- Thus, rather than having a global set of predicates, we can have a *different predicate set* for each *different location*
 - Since the model checker is very *sensitive* to the *number* of predicates, this is really important for scalability

- Summary: (*CEGAR*)



- Can both verify and give counterexamples
- but no termination guarantees...

- 作业 1
- 作业 2: 分别使用interval domain下的Least Fixed Point算法、Widening算法和Narrowing算法求出模型中 y 在各点的估计。
- 作业 3
- 作业 4

本章节大作业参考论文

大作业可参考论文 (但不限于下列论文):

- 经典

- Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints

- 应用

- AI²: Safety and Robustness Certification of Neural Networks with Abstract Interpretation
- Extracting Protocol Format as State Machine via Controlled Static Loop Analysis
- Rule-Based Static Analysis of Network Protocol Implementations
- Precise Enforcement of Progress-Sensitive Security