Half Lecture 13 (or-so)
Predicate Abstraction and Intervals

2015

Predicate Abstraction

Abstract interpretation domain (lattice) is determined by a set of formulas (predicates) $\mathcal{P}$ on program variables.
Example: $\mathcal{P} = \{P_0, P_1, P_2, P_3\}$ where

$$\begin{array}{rcl} P_0 & \equiv & \text{false} \\ P_1 & \equiv & 0 < x \\ P_2 & \equiv & 0 < y \\ P_3 & \equiv & x < y \end{array}$$

Analysis tries to construct invariants from these predicates using

- conjunctions, e.g. $P_1 \wedge P_3$ (our focus here, for simplicity)
- conjunctions and disjunctions, e.g. $P_3 \wedge (P_1 \vee P_2)$

Predicate Abstraction

Abstract interpretation domain (lattice) is determined by a set of formulas (predicates) $\mathcal{P}$ on program variables.
Example: $\mathcal{P} = \{P_0, P_1, P_2, P_3\}$ where

$$\begin{array}{lcl} P_0 & \equiv & \text{false} \\ P_1 & \equiv & 0 < x \\ P_2 & \equiv & 0 < y \\ P_3 & \equiv & x < y \end{array}$$

Analysis tries to construct invariants from these predicates using

- conjunctions, e.g. $P_1 \wedge P_3$ (our focus here, for simplicity)
- conjunctions and disjunctions, e.g. $P_3 \wedge (P_1 \vee P_2)$

We assume $P_0 \equiv$ false, other predicates in $\mathcal{P}$ - arbitrary

- expressed in logic of some theorem prover (e.g. SMT solver)

Example of Analysis Result

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 <= x, 0 < y, x < y, x = 0, y = 1, x < 1000, 1000 \leq x\}$

```
x = 0;
y = 1;
// 0<y, x<y,x=0,y=1, x<1000
while // 0<y, 0≤x, x<y
(x < 1000) {
  // 0<y, 0≤x, x<y, x<1000
  x = x + 1;
  // 0<y, 0≤x, 0<x
  y = 2*x;
  // 0<y, 0≤x, 0<x, x<y
  y = y + 1;
  // 0<y, 0≤x, 0<x, x<y
  print(y);
}
// 0<y, 0≤x, x<y, 1000 ≤ x
```

Example of Analysis Result

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 <= x, 0 < y, x < y, x = 0, y = 1, x < 1000, 1000 \leq x\}$

```
x = 0;
y = 1;
// 0<y, x<y,x=0,y=1, x<1000
while // 0<y, 0≤x, x<y
(x < 1000) {
  // 0<y, 0≤x, x<y, x<1000
  x = x + 1;
  // 0<y, 0≤x, 0<x
  y = 2*x;
  // 0<y, 0≤x, 0<x, x<y
  y = y + 1;
  // 0<y, 0≤x, 0<x, x<y
  print(y);
}
// 0<y, 0≤x, x<y, 1000 ≤ x
```

Start by assuming all predicates hold in all non-entry points.

Example of Analysis Result

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 <= x, 0 < y, x < y, x = 0, y = 1, x < 1000, 1000 \leq x\}$

x = 0;
y = 1;
// 0<y, x<y,x=0,y=1, x<1000
while // 0<y, 0≤x, x<y
(x < 1000) {
  // 0<y, 0≤x, x<y, x<1000
  x = x + 1;
  // 0<y, 0≤x, 0<x
  y = 2∗x;
  // 0<y, 0≤x, 0<x, x<y
  y = y + 1;
  // 0<y, 0≤x, 0<x, x<y
  print(y);
}
// 0<y, 0≤x, x<y, 1000 ≤ x

Start by assuming all predicates hold in all non-entry points.

Check Hoare triples, remove predicates from postcondition that do not hold

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) =$

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \wedge x < y\}$.

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \wedge x < y\}$. We often assume states are pairs (x, y). Then $\gamma(a_0) =$

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \wedge x < y\}$. We often assume states are pairs (x, y). Then $\gamma(a_0) = \{(x, y) \mid 0 < x \wedge x < y\}$.

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \wedge x < y\}$. We often assume states are pairs (x, y). Then $\gamma(a_0) = \{(x, y) \mid 0 < x \wedge x < y\}$.
If $a_1 \subseteq a_2$ then $\bigwedge a_2$ implies $\bigwedge a_1$, so $\gamma(a_2) \subseteq \gamma(a_1)$.

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \wedge x < y\}$. We often assume states are pairs (x, y). Then $\gamma(a_0) = \{(x, y) \mid 0 < x \wedge x < y\}$.
If $a_1 \subseteq a_2$ then $\bigwedge a_2$ implies $\bigwedge a_1$, so $\gamma(a_2) \subseteq \gamma(a_1)$.
Define:

$$a_1 \sqsubseteq a_2 \quad \iff \quad a_2 \subseteq a_1$$

Lemma: $a_1 \sqsubseteq a_2 \rightarrow \gamma(a_1) \subseteq \gamma(a_2)$

Lattice of Conjunctions of Predicates and Concretization

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$ - predicates

- formulas whose free variables denote program variables

$L = A = 2^{\mathcal{P}}$, so for $a \in A$ we have $a \subseteq \mathcal{P}$
Example: $a_0 = \{0 < x, x < y\}$.
$s \models F$ means: formula F is true for variables given by the program state s

$$\gamma(a) = \{s \mid s \models \bigwedge_{P \in a} P\}$$

Shorthand: $\bigwedge a$ means $\bigwedge_{P \in a} P$
Example: $\gamma(a_0) = \{s \mid s \models 0 < x \land x < y\}$. We often assume states are pairs (x, y). Then $\gamma(a_0) = \{(x, y) \mid 0 < x \land x < y\}$.
If $a_1 \subseteq a_2$ then $\bigwedge a_2$ implies $\bigwedge a_1$, so $\gamma(a_2) \subseteq \gamma(a_1)$.
Define:

$$a_1 \sqsubseteq a_2 \quad \iff \quad a_2 \subseteq a_1$$

Lemma: $a_1 \sqsubseteq a_2 \to \gamma(a_1) \subseteq \gamma(a_2)$
Does the converse hold?

Lattice Operations: Example

$$\{\mathit{false}, 0 < x, x < y\} \sqsubseteq \{0 < x, 0 < y\} \sqsubseteq \{0 < x\} \sqsubseteq \emptyset$$

Draw the Hasse diagram for the lattice $(A, \sqsubseteq)$ i.e. $(2^{\mathcal{P}}, \supseteq)$ for $\mathcal{P} = \{P_0, P_1, P_2\}$ a three-element set.

Lattice Operations: Example

$$\{\mathit{false}, 0 < x, x < y\} \sqsubseteq \{0 < x, 0 < y\} \sqsubseteq \{0 < x\} \sqsubseteq \emptyset$$

Draw the Hasse diagram for the lattice $(A, \sqsubseteq)$ i.e. $(2^{\mathcal{P}}, \supseteq)$ for $\mathcal{P} = \{P_0, P_1, P_2\}$ a three-element set.

What is the top and what is the bottom element of this lattice?

Lattice Operations: Example

$$\{\mathit{false}, 0 < x, x < y\} \sqsubseteq \{0 < x, 0 < y\} \sqsubseteq \{0 < x\} \sqsubseteq \emptyset$$

Draw the Hasse diagram for the lattice $(A, \sqsubseteq)$ i.e. $(2^{\mathcal{P}}, \supseteq)$ for $\mathcal{P} = \{P_0, P_1, P_2\}$ a three-element set.

What is the top and what is the bottom element of this lattice?
What is $\sqcup$? Compute $\{0 < x, x < y\} \sqcup \{0 < y, x < y\} =$

Lattice Operations: Example

$$\{\mathit{false}, 0 < x, x < y\} \sqsubseteq \{0 < x, 0 < y\} \sqsubseteq \{0 < x\} \sqsubseteq \emptyset$$

Draw the Hasse diagram for the lattice $(A, \sqsubseteq)$ i.e. $(2^{\mathcal{P}}, \supseteq)$ for $\mathcal{P} = \{P_0, P_1, P_2\}$ a three-element set.

What is the top and what is the bottom element of this lattice?
What is $\sqcup$? Compute $\{0 < x, x < y\} \sqcup \{0 < y, x < y\} = \{x < y\}$ (draw)

Lattice Operations: Example

$$\{\mathit{false}, 0 < x, x < y\} \sqsubseteq \{0 < x, 0 < y\} \sqsubseteq \{0 < x\} \sqsubseteq \emptyset$$

Draw the Hasse diagram for the lattice $(A, \sqsubseteq)$ i.e. $(2^{\mathcal{P}}, \supseteq)$ for $\mathcal{P} = \{P_0, P_1, P_2\}$ a three-element set.

What is the top and what is the bottom element of this lattice?
What is $\sqcup$? Compute $\{0 < x, x < y\} \sqcup \{0 < y, x < y\} = \{x < y\}$ (draw)
What is the size and the height of the lattice?

Lattice Height

A finite chain inside a partial order is a strictly ordered sequence of elements: $x_0 \sqsubset x_1 \ldots \sqsubset x_n$
Here $x \sqsubset y$ means that both $x \sqsubseteq y$ and $x \neq y$
Length of such chain is n (number of $\sqsubset$ signs)
Note that $x_i \sqsubseteq x_j$ for $i < j$ by transitivity
Thus all elements in a chain are distinct.
An infinite chain is infinite sequence of elements where $x_i \sqsubset x_{i+1}$ for all i
A lattice is finite-height if all chains are finite. Then the maximum length of chains is called the height of the lattice.

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is:

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is:

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is:

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\bot$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$
- Size of the lattice with $n+1$ element:

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \dots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$
- Size of the lattice with $n+1$ element: 2^{n+1}

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\bot$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$
- Size of the lattice with $n+1$ element: 2^{n+1}
- Height of the lattice with $n+1$ element:

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\perp$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$
- Size of the lattice with $n+1$ element: 2^{n+1}
- Height of the lattice with $n+1$ element: $n+1$

Lattice of Predicates

$\mathcal{P} = \{P_0, P_1, \ldots, P_n\}$. Lattice elements $a \in 2^{\mathcal{P}}$ (subsets of $\mathcal{P}$)
More predicates in conjunction $\Rightarrow$ stronger condition $\Rightarrow$ smaller set
Therefore we have:

- $\bot$ - bottom (smallest set of states) is: $\mathcal{P}$
- $\top$ - top (largest set of states) is: $\emptyset$ (predicate 'true')
- $\sqcup$ (approximates union) is: $\cap$
- Size of the lattice with $n+1$ element: 2^{n+1}
- Height of the lattice with $n+1$ element: $n+1$

Given $a \in 2^{\mathcal{P}}$ we abbreviate $\bigwedge_{P \in a} P$ as $\bigwedge a$

Abstract Strongest Postcondition

Abstract strongest postcondition (= transfer function in data-flow analysis)
Consider a command c and a set of predicates $a \subseteq \mathcal{P}$, we define *abstract* strongest postcondition of a as the conjunction of all predicates from $\mathcal{P}$ that hold after c:

$$sp^{\#}(a) = \big\{ P' \in \mathcal{P} \mid \ \{\bigwedge a\} c \{P'\} \ \big\}$$

Note that $\{\ldots\}c\{\ldots\}$ after "|" denotes a Hoare triple
By conjunctivity of Hoare triple, the result denotes a valid postcondition:

$$\{\bigwedge a\} c \{\bigwedge sp^{\#}(a)\}$$

Thus $sp_F(\bigwedge a, c) \implies sp^{\#}(a)$ holds as sp_F is strongest. However, converse implication need not - abstract postcondition is only an over-approximation
This definition of $sp^{\#}(a)$ gives the strongest condition *that we can write* as a conjunction of the allowed predicates $\mathcal{P}$, whereas sp_F need not be expressible using $\mathcal{P}$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \wedge (y' = x + 1 \wedge x' = x) \implies P'(x', y')$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \land (y' = x + 1 \land x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \land (y' = x + 1 \land x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$sp^{\#}(\{0 < x\}, y := x - 1) =$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \land (y' = x + 1 \land x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$sp^{\#}(\{0 < x\}, y := x - 1) = \{0 < x\}$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \land (y' = x + 1 \land x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$sp^{\#}(\{0 < x\}, y := x - 1) = \{0 < x\}$$
$$sp^{\#}(\{0 < x, x < y\}, x := x - 1) =$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $\mathit{sp}^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \wedge (y' = x + 1 \wedge x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$\mathit{sp}^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$\mathit{sp}^{\#}(\{0 < x\}, y := x - 1) = \{0 < x\}$$
$$\mathit{sp}^{\#}(\{0 < x, x < y\}, x := x - 1) = \{0 < y, x < y\}$$

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \wedge (y' = x + 1 \wedge x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$sp^{\#}(\{0 < x\}, y := x - 1) = \{0 < x\}$$
$$sp^{\#}(\{0 < x, x < y\}, x := x - 1) = \{0 < y, x < y\}$$

What is the relation between $\{0 < x, x < y\}$ and $\{0 < x, 0 < y, x < y\}$?

Example of Computing Abstract Strongest Postcondition

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 < y, x < y\}$
Compute $sp^{\#}(\{0 < x\}, y := x + 1)$
We can test for each predicate $P' \in \mathcal{P}$ whether

$$x > 0 \wedge (y' = x + 1 \wedge x' = x) \implies P'(x', y')$$

We obtain that the condition holds for $0 < x$, $0 < y$, and for $x < y$, but not for *false*. Thus,

$$sp^{\#}(\{0 < x\}, y := x + 1) = \{0 < x, 0 < y, x < y\}$$

Compute

$$sp^{\#}(\{0 < x\}, y := x - 1) = \{0 < x\}$$
$$sp^{\#}(\{0 < x, x < y\}, x := x - 1) = \{0 < y, x < y\}$$

What is the relation between $\{0 < x, x < y\}$ and $\{0 < x, 0 < y, x < y\}$?
Different in lattice, denote same states. This is not a problem.

Analysis Algorithm

Given control-flow graph (V, E) where E contains triples (u, c, v) where c is a command labeling the edge

Analysis maintains a map $g : V \to A$ from vertices to lattice elements

For program entry point: the set of all predicates that are true in every initial state (over-approximation of the set of initial states).
For other program points, initially put conjunction of all predicates (bottom). Then repeatedly update the value at a point when some predecessor changes:

$$g(v) := \bigsqcup_{(u,c,v) \in E} sp^{\#}(g(u), c)$$

This process terminates since the lattice has finite height. Lattice elements grow (sets of predicates shrink).
Checking if the process terminates is same as checking that we have computed a loop invariant.

Running the Example from the Initial State

$\mathcal{P} = \{\mathit{false}, 0 < x, 0 <= x, 0 < y, x < y, x = 0, y = 1, x < 1000, 1000 \leq x\}$

```
// true
x = 0;
// false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
y = 1;
while // false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
(x < 1000) {
   // false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
   x = x + 1;
   // false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
   y = 2*x;
   // false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
   y = y + 1;
   // false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
}
// false, 0<x, 0≤x, 0<y, x<y, x=0, y=1, x<1000,1000≤x
```

Example of Limitations of Conjunctions

$\mathcal{P} = \{false, 0 < x, x \leq 0, 0 < y\}$

```
if (x > 0) {
  y = x
}
// Q
if (x > 0) {
  if(y > 0) 1/x
  else error
}
```

Assuming arbitrary initial state, what is the best we can compute as Q using conjunctions from $\mathcal{P}$?

Example of Limitations of Conjunctions

$\mathcal{P} = \{false, 0 < x, x \leq 0, 0 < y\}$

if (x $>$ 0) $\{$
  y $=$ x
$\}$
// Q
if (x $>$ 0) $\{$
  if(y $>$ 0) 1/x
  else error
$\}$

Assuming arbitrary initial state, what is the best we can compute as Q using conjunctions from $\mathcal{P}$? 'true'

Using disjunctions of conjunctions:

Example of Limitations of Conjunctions

$\mathcal{P} = \{false, 0 < x, x \leq 0, 0 < y\}$

if $(x > 0)$ $\{$
  $y = x$
$\}$
// Q
if $(x > 0)$ $\{$
  if$(y > 0)$ $1/x$
  else error
$\}$

Assuming arbitrary initial state, what is the best we can compute as Q using conjunctions from $\mathcal{P}$? 'true'

Using disjunctions of conjunctions: $(x > 0 \wedge y > 0) \vee (x \leq 0)$
Allows us to prove absence of error in the remaining code

Disjunctive Analysis to Overcome Limitations

Lattice with *disjunction* of conjunctions

- Sets of sets of predicates - exponentially larger
- Reduce by using as few predicates as possible, different possible predicates for each program point, limit sizes of conjuncts, ...

Important topic: automatically discover predicates.

- in general as hard as discovering loop invariants
- yet we only need to discover pieces of invariants
- and we can conservatively suggest more candidate predicates (any predicate set gives a sound analysis)

Interval Analysis

For a machine integer x, compute the interval $[a, b]$ such that $x \in [a, b]$
Worst-case interval: $[minI, maxI] = [-2^{31}, 2^{31} - 1]$

- each machine integer is between smallest and largest representable one

In addition, we introduce a special $\perp$ interval to represent an empty set of states
Consider relation c whose semantics is relation r on initial and final integer
Define $sp^{\#}$ as the interval for the values that x can take after c:

$$sp^{\#}([a, b], r) = \alpha(\{x' \mid x \in [a, b] \wedge (x, x') \in r\})$$

Here α computes the interval for a set of values:

- $\alpha(S) = [\min(S), \max(S)]$, if $S \neq \emptyset$, whereas $\alpha(\emptyset) = \perp$

We define $sp^{\#}(\perp) = \perp$, since image of empty set is an empty set

$sp^{\#}([0, 10], x{=}x + 7) = [7, 17]$ $\quad sp^{\#}([-5, -5], x{=}x * x) = [0, 25]$
$sp^{\#}([1000, maxI], x = x + 30) = [minI, maxI]$

Size of the Interval Lattice

$L = \{\perp\} \cup \{[a, b] \mid minI \leq a \leq b \leq maxI\}$
Here $\perp \sqsubseteq [a, b]$ for all proper intervals $[a, b]$. Between intervals,

$$[a, b] \sqsubseteq [a', b'] \text{ if and only if } a' \leq a \leq b \leq b'$$

Number of elements in the lattice:

Size of the Interval Lattice

$L = \{\perp\} \cup \{[a, b] \mid minI \leq a \leq b \leq maxI\}$
Here $\perp \sqsubseteq [a, b]$ for all proper intervals $[a, b]$. Between intervals,

$$[a, b] \sqsubseteq [a', b'] \text{ if and only if } a' \leq a \leq b \leq b'$$

Number of elements in the lattice:$1 + \frac{2^{32}(2^{32}+1)}{2} = 1 + 2^{31} + 2^{63}$

Size of the Interval Lattice

$L = \{\bot\} \cup \{[a, b] \mid minI \leq a \leq b \leq maxI\}$
Here $\bot \sqsubseteq [a, b]$ for all proper intervals $[a, b]$. Between intervals,

$$[a, b] \sqsubseteq [a', b'] \text{ if and only if } a' \leq a \leq b \leq b'$$

Number of elements in the lattice: $1 + \frac{2^{32}(2^{32}+1)}{2} = 1 + 2^{31} + 2^{63}$
Size of the longest chain:

Size of the Interval Lattice

$L = \{\perp\} \cup \{[a, b] \mid minI \leq a \leq b \leq maxI\}$
Here $\perp \sqsubseteq [a, b]$ for all proper intervals $[a, b]$. Between intervals,

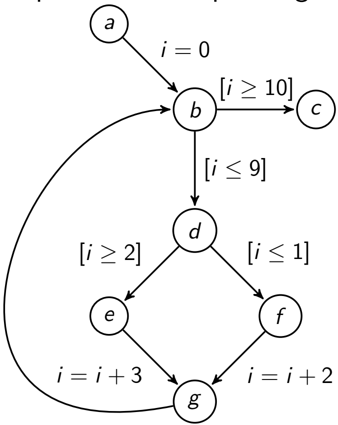
$$[a, b] \sqsubseteq [a', b'] \text{ if and only if } a' \leq a \leq b \leq b'$$

Number of elements in the lattice: $1 + \frac{2^{32}(2^{32}+1)}{2} = 1 + 2^{31} + 2^{63}$
Size of the longest chain: 2^{32}

Example

```
//a
i = 0;
    //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

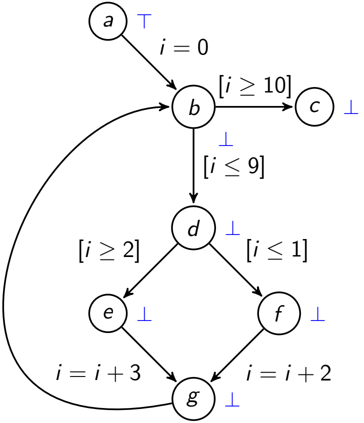
One possible corresponding control-flow graph is:



Starting Point for Analysis

```
//a
i = 0;
      //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

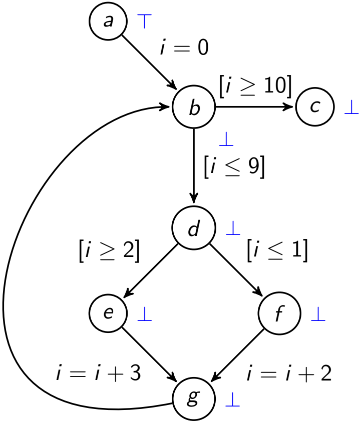
One possible corresponding control-flow graph is:



Starting Point for Analysis

```
//a
i = 0;
      //b
while (i < 10) {
  //d
  if (i > 1)
     //e
     i = i + 3;
  else
     //f
     i = i + 2;
  //g
}
//c
```

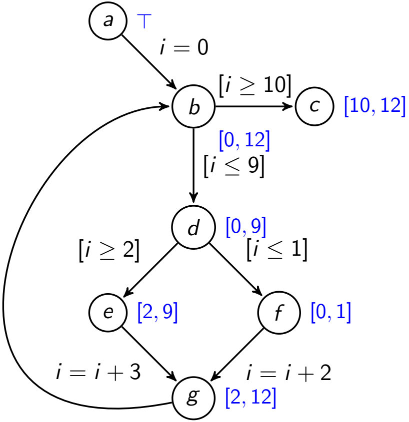
One possible corresponding control-flow graph is:



Fixpoint Found

```
//a
i = 0;
    //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

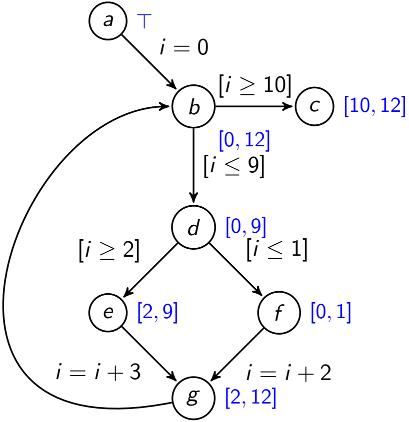
One possible corresponding control-flow graph is:



Fixpoint Found

```
//a
i = 0;
    //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

One possible corresponding control-flow graph is:



Note: in general, we maintain interval for each variable

General Remarks

Whatever we choose as our abstract domain A (typically some lattice), it is good to have a function γ that gives meaning to elements of A
Often, elements $a \in A$ represent sets of states:

- predicate abstraction: states that satisfy the conjunction of predicates
- interval analysis: states whose variables belong to the intervals

When we think about correctness conditions intuitively, we can "almost ignore" γ (but it is needed for statements to type check).
Each analysis is given by transfer functions such as $sp^{\#}$ which need to satisfy, for each $a \in A$:

$$sp(c, \gamma(a)) \subseteq \gamma(sp^{\#}(c, a))$$

$sp^{\#}$ gives a larger set of states (it finds only some, not all properties)

computed properties imply assertions of interest $\Rightarrow$ we proved the assertions
otherwise $\Rightarrow$ either assertions do not hold, or analysis was too conservative