Iterative Program Analysis
Abstract Interpretation

Mooly Sagiv

Textbook: Principles of Program Analysis

Chapter 4

POPL 79, 92 Cousot & Cousot

Outline

- The abstract interpretation technique
  - The main theorem
  - Applications
  - Precision
  - Complexity
  - Widening
- Later on
  - Combining Analysis
  - Interprocedural Analysis
  - Shape Analysis

Complete Lattices

- A poset $(L, \sqsubseteq)$ is a complete lattice if every subset has least and upper bounds
- $L = (L, \sqsubseteq) = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
  - $\bot = \bigsqcup \emptyset = \bigsqcap L$
  - $\top = \bigsqcup L = \bigsqcap \emptyset$
- Examples
  - Total orders $(N, \leq)$
  - Powersets $(P(S), \subseteq)$
  - Powersets $(P(S), \supseteq)$
  - Constant propagation

Soundness Theorem(1)

1. Let (α, γ) form Galois connection from C to A
2. $f: C \to C$ be a monotone function
3. $f^{\#}: A \to A$ be a monotone function
4. $\forall a \in A: f(\gamma(a)) \sqsubseteq \gamma(f^{\#}(a))$

$$lfp(f) \sqsubseteq \gamma(lfp(f^{\#}))$$

$$\alpha(lfp(f)) \sqsubseteq lfp(f^{\#})$$

Soundness Theorem(2)

1. Let (α, γ) form Galois connection from C to A
2. $f: C \rightarrow C$ be a monotone function
3. $f^{\#} : A \rightarrow A$ be a monotone function
4. $\forall c \in C: \alpha(f(c)) \sqsubseteq f^{\#}(\alpha(c))$

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^{\#})$$

$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^{\#}))$$

Soundness Theorem(3)

1. Let (α, γ) form Galois connection from C to A
2. $f: C \rightarrow C$ be a monotone function
3. $f^{\#}: A \rightarrow A$ be a monotone function
4. $\forall a \in A: \alpha(f(\gamma(a))) \sqsubseteq f^{\#}(a)$

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^{\#})$$

$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^{\#}))$$

Completeness

$$\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}(f^{\#})$$

$$\mathrm{lfp}(f) = \gamma(\mathrm{lfp}(f^{\#}))$$

Flowchart Programs

- A finite control flow graph $G(s, N, A)$ where
  - $A \subseteq N \times N$
  - $s \in N$ is the start vertex, s has no incoming arcs and there is a path from s to every vertex
  - A special node error $\in N$
- Every arc in A is labeled with two types of operations
  - assume e;
  - e := e';

Simple Example

z := 3;

x := 1

while x > 0 {

  if x == 4 then

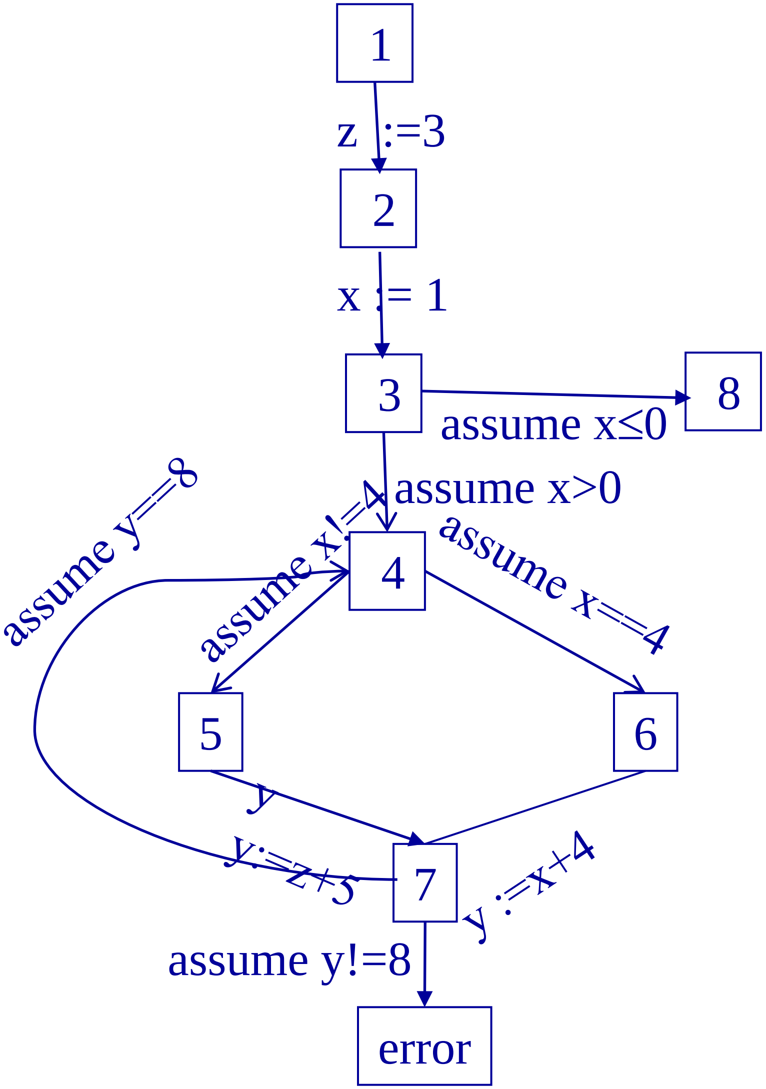
      y := x+ 4;

  else

    y := z + 5;

  assert y == 8;

 }



Collecting (Concrete)Interpretation of Flowchart Programs

- A set of (usually infinite) states Σ
- A lattice $<P(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma>$
- A set of initial states at s: $\Sigma_s \subseteq \Sigma$
- The semantics of operations on arcs
  - $[\![\text{assume } e]\!] = \{<\sigma, \sigma> \mid \sigma \models e\}$
  - $[\![x := e]\!] = \{<\sigma, \sigma'> \mid \sigma' = \sigma[x \mapsto [\![e]\!](\sigma)\}$
- The collecting interpretation is the least fixed point of the following system:
  - $CS[s] = \Sigma_s$
  - $CS[n] = \cup \{\sigma' \mid <m, n> \in E, \sigma \in CS[m], <\sigma, \sigma'> \in [\![<m, n>]\!]\}$ for $n \neq s$

Constant Propagation

- A lattice $< [Var \rightarrow Z \cup \{\top, \bot\}], \sqsubseteq, \sqcup, \sqcap, \bot, \top >$
- β: $[Var \rightarrow Z] \rightarrow [Var \rightarrow Z \cup \{\top, \bot\}]$
  - $\beta(\sigma) = (\sigma)$
- α: $P([Var \rightarrow Z]) \rightarrow [Var \rightarrow Z \cup \{\top, \bot\}]$
  - $\alpha(X) = \sqcup \{\beta(\sigma) \mid \sigma \in X\} = \sqcup \{\sigma \mid \sigma \in X\}$
- γ:$[Var \rightarrow Z \cup \{\top, \bot\}] \rightarrow P([Var \rightarrow Z])$
  - $\gamma(\sigma^{\#}) = \{\sigma \mid \beta(\sigma) \sqsubseteq \sigma^{\#} \} = \{\sigma \mid \sigma \sqsubseteq \sigma^{\#} \}$
- Initial value at s
- The semantics of operations on arcs
  - $[\![\text{assume } x == c]\!]^{\#}$
  - $[\![x := e]\!]^{\#} =$
- Local Soundness and optimality

Example: May-Be-Garbage

- A variable x may-be-garbage at a program point v if there exists a execution path leading to v in which x's value is unpredictable:
  - Was not assigned
  - Was assigned using an unpredictable expression
- Lattice
- Galois connection
- Basic statements
- Soundness

Pointer Language

$a ::= x \mid *x \mid \&x \mid \ldots$

b ::= true | a = a| not b

assume b

$x := a$

$*x := y$

Collecting Semantics for Pointers

$\text{State1} = [\text{Loc} \rightarrow \text{Loc} \cup Z]$

Points-To Analysis

- Lattice L_{pt} =
- Galois connection
- Meaning of statements

```
t := &a;
y := &b;

z := &c;

 if x> 0
         then p:= &y;
         else p:= &z;

 *p := t;
```

```
/*  ∅ */ t := &a; /* {(t, a)}*/
 /* {(t, a)}*/  y := &b; /* {(t, a), (y, b) }*/

/* {(t, a), (y, b)}*/ z := &c; /* {(t, a), (y, b), (z, c) }*/

 if x> 0;
         then p:= &y; /* {(t, a),  (y, b), (z, c), (p, y)}*/


         else p:= &z; /* {(t, a),  (y, b), (z, c), (p, z)}*/
/* {(t, a),  (y, b), (z, c), (p, y), (p, z)}*/

 *p := t;

/* {(t, a), (y, b), (y, c), (p, y), (p, z), (y, a), (z, a)}*/
```

Abstract Transformers

$State^{\#} = P(Var^* \times Var^*)$

$[\![x := a]\!]^{\#}$

$[\![x := \&y]\!]^{\#}$

$[\![x := *y]\!]^{\#}$

$[\![x := y]\!]^{\#}$

$[\![*x := y]\!]^{\#}$

$[\![\text{assume } x == y]\!]^{\#}$

$[\![\text{assume } x \mathrel{!=} y]\!]^{\#}$

```
/*  ∅ */ t := &a; /* {(t, a)}*/
 /* {(t, a)}*/  y := &b; /* {(t, a), (y, b) }*/

/* {(t, a), (y, b)}*/ z := &c; /* {(t, a), (y, b), (z, c) }*/

 if x> 0;
        then p:= &y; /* {(t, a),  (y, b), (z, c), (p, y)}*/


        else p:= &z; /* {(t, a),  (y, b), (z, c), (p, z)}*/
/* {(t, a),  (y, b), (z, c), (p, y), (p, z)}*/

 *p := t;

/* {(t, a), (y, b), (y, c), (p, y), (p, z), (y, a), (z, a)}*/
```

Flow insensitive points-to-analysis Steengard 1996

- Ignore control flow
- One set of points-to per program
- Can be represented as a directed graph
- Conservative approximation
  - Accumulate pointers
- Can be computed in almost linear time
  - Union find

```
t := &a;
y := &b;

z := &c;

 if x> 0;
         then p:= &y;
         else p:= &z;

 *p := t;
```

Precision

- We cannot usually have
  - $\alpha(CS) = DF$
    on all programs
- But can we say something about precision in all programs?

The Join-Over-All-Paths (JOP)

- Let paths(v) denote the potentially infinite set paths from start to v (written as sequences of edges)
- For a sequence of edges $[e_1, e_2, \ldots, e_n]$ define $f^{\#}[e_1, e_2, \ldots, e_n]: L \to L$ by composing the effects of basic blocks
  $f^{\#}[e_1, e_2, \ldots, e_n](l) = f^{\#}(e_n)(\ldots (f^{\#}(e_2)(f^{\#}(e_1)\ (l))\ \ldots)$
- JOP[v] $= \sqcup\{f^{\#}[e_1, e_2, \ldots,e_n](\iota)$
  $[e_1, e_2, \ldots, e_n] \in \text{paths}(v)\}$

JOP vs. Least Solution

- The df solution obtained by Chaotic iteration satisfies for every *v*:
  - $JOP[v] \sqsubseteq df(v)$
- A function $f^{\#}$ is additive (distributive) if
  - $f^{\#}(\sqcup\{x \mid x \in X\}) = \sqcup\{f^{\#}(x) \mid \in X\}$
- If every $f^{\#}_{(u,v)}$ is additive (distributive) for all the edges (u,v)
  - $JOP[v] = df(v)$
- Examples
  - Maybe garbage
  - Constant Propagation
  - Points-to

Notions of precision

- $CS = \gamma$ (df)
- $\alpha(CS) =$ df
- Meet(Join) over all paths
- Using best transformers
- Good enough

Complexity of Chaotic Iterations

- Usually depends on the height of the lattice
- In some cases better bound exist
- A function f is fast if $f(f(l)) \sqsubseteq l \sqcup f(l)$
- For fast functions the Chaotic iterations can be implemented in O(nest * |V|) iterations
  - nest is the number of nested loop
  - |V| is the number of control flow nodes

Conclusion

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- But expensive
  - More efficient methods exist for structured programs
- Abstract interpretation relates runtime semantics and static information
- The concrete semantics serves as a tool in designing abstractions