Iterative Program Analysis
Abstract Interpretation

Mooly Sagiv
http://www.cs.tau.ac.il/~msagiv/courses/pa11.html
Tel Aviv University
640-6706

Textbook: Principles of Program Analysis
Chapter 4
CC79, CC92

Outline

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

Soundness Theorem(1)

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. $\forall a \in A: f(\gamma(a)) \sqsubseteq \gamma(f^{\#}(a))$

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

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

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. $\forall c \in C: \alpha(f(c)) \sqsubseteq f^{\#}(\alpha(c))$

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

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

Soundness Theorem(3)

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. $\forall a \in A: \alpha(f(\gamma(a))) \sqsubseteq f^{\#}(a)$

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

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

Completeness

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

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

Constant Propagation

- β: [Var $\rightarrow$ Z] $\rightarrow$ [Var $\rightarrow$ Z$\cup${τ, $\perp$}]
  - $\beta(\sigma) = (\sigma)$
- α: P([Var $\rightarrow$ Z]) $\rightarrow$ [Var $\rightarrow$ Z$\cup${τ, $\perp$}]
  - $\alpha(X) = \sqcup \{\beta(\sigma) \mid \sigma \in X\} = \sqcup \{\sigma \mid \sigma \in X\}$
- γ:[Var $\rightarrow$ Z $\cup${τ, $\perp$}] $\rightarrow$ P([Var $\rightarrow$ Z])
  - $\gamma(\sigma^{\#}) = \{\sigma \mid \beta(\sigma) \sqsubseteq \sigma^{\#}\} = \{\sigma \mid \sigma \sqsubseteq \sigma^{\#}\}$
- Local Soundness
  - $[\![st]\!]^{\#}(\sigma^{\#}) \sqsupseteq \alpha(\{ [\![st]\!] \sigma \mid \sigma \in \gamma(\sigma^{\#}) = \sqcup \{ [\![st]\!] \sigma \mid \sigma \sqsubseteq \sigma^{\#} \}$
- Optimality (Induced)
  - $[\![st]\!]^{\#}(\sigma^{\#}) = \alpha(\{ [\![st]\!] \sigma \mid \sigma \in \gamma(\sigma^{\#})\} = \sqcup \{ [\![st]\!] \sigma \mid \sigma \sqsubseteq \sigma^{\#} \}$
- Soundness
- Completeness

Formal available expression

- Find out which expressions are available at a given program point
- Example program

```
x = y + t  //{(y+t)}
z = y + r  //{(y+t), (y+r)}
while (…) { // {(y+t), (y+r)}
        t = t + (y + r)  // {(y+t), (y+r), t + (y +r )}
                }
```

Available expression lattice

- P(Fexp)
- $X \sqsubseteq Y \Leftrightarrow X \supseteq Y$
- $X \sqcup Y = X \cap Y$
- $\bot = Fexp$
- $\top = \varnothing$

Available expressions

- Computing Fexp for while

  s ::= skip { s.Fexp := ∅ }

  s := id = exp { s.Fexp = {exp.rep }

  s := s ; s { s.Fexp := s[1].Fexp ∪ s[2].Fexp }

  s := while exp do s { s.Fexp := {exp.rep} ∪ s[1].Fexp }

  s := if exp then s else s

  { s.Fexp := {exp.rep} ∪ s[1].Fexp ∪ s[2].Fexp }

Instrumented Semantics
Available expressions

- $S[\![Stm]\!] : State \times P(Fexp) \rightarrow State \times P(Fexp)$
- $S [\![id := a]\!] (\sigma, ae) = (\sigma[a \mapsto a [\![a]\!] \sigma], notArg(id, ae \cup \{a\})$
- $S [\![skip]\!] (\sigma, ae) = (\sigma, ae)$

- $CS[\![Stm]\!] : P(State \times P(Fexp)) \rightarrow P(State \times P(Fexp))$
- $CS [\![s]\!] (X) = \{S [\![s]\!] (\sigma, ae) \mid (\sigma, ae) \in X\}$

Collecting Semantics Example

```
x  = y  * z;
if  (x == 6)
        y = z+ t;
…
if  (x == 6)
        r = z + t;
```

Formal Available Expressions Abstraction

- β: $[Var \rightarrow Z] \times P(Fexp) \rightarrow P(Fexp)$
  - $\beta(\sigma, ae) = (ae)$
- α: $P(P([Var \rightarrow Z] \times P(Fexp)) \rightarrow P(Fexp)$
  - $\alpha(X) = \sqcup \{\beta(\sigma) \mid \sigma \in X\} = \cap\{ae \mid (\sigma, ae) \in X\}$
- γ: $P(Fexp) \rightarrow P(P([Var \rightarrow Z] \times P(Fexp))$
  - $\gamma(ae^{\#}) = \{(\sigma, ae) \mid \beta(\sigma, ae) \sqsubseteq ae^{\#} \} = \{(\sigma, ae) \mid ae \supseteq ae^{\#} \}$

Formal Available Expressions AI

- $S[\![Stm]\!]^{\#} : P(Fexp) \rightarrow P(Fexp)$
- $S[\![id := a]\!]^{\#} (ae) = notArg(id, ae \cup \{a\})$
- $S[\![skip]\!]^{\#} (ae) = (ae)$
- Local Soundness
  - $[\![st]\!]^{\#}(ae^{\#}) \sqsupseteq \sqcup \{ ([\![st]\!] (\sigma, ae)[1] \mid ae \sqsubseteq ae^{\#} \}$
- Optimality
  - $[\![st]\!]^{\#}(ae^{\#}) = \alpha(\{ [\![st]\!] (\sigma, ae) \mid (\sigma, ae) \in \gamma (\sigma^{\#})\} = \sqcup \{ ([\![st]\!] (\sigma, ae)[1] \mid ae \sqsubseteq ae^{\#} \}$
- The initial value at the entry is $\varnothing$
- Example program

```
x = y + t  //{(y+t)}
z = y + r  //{(y+t), (y+r)}
while (…) { // {(y+t), (y+r)}
      t = t + (y + r)  // { (y+r)}
            }
```

- Soundness and Completeness

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

The **PWhile** Programming Language
Abstract Syntax

$a := x \mid *x \mid \&x \mid n \mid a_1\ op_a\ a_2$

$b :=$ true | false | not b | $b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$

$S := x := a \mid *x := a \mid$ skip | $S_1 ; S_2 \mid$
if b **then** S_1 **else** S_2 | **while** b do S

Concrete Semantics for PWhile

State1= [Loc$\rightarrow$Loc$\cup$Z]

For every atomic statement S

$[\![S]\!]$: States1 $\rightarrow$States1

$[\![x := a]\!](\sigma)=\sigma[loc(x) \mapsto \mathbf{A}[\![a]\!]\ \sigma]$

$[\![x := \&y]\!](\sigma)$

$[\![x := *y]\!](\sigma)$

$[\![x := y]\!](\sigma)$

$[\![*x := y]\!](\sigma)$

Points-To Analysis

- Lattice L_{pt} =
- Galois connection

```
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 Semantics for PWhile

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

For every atomic statement S

$[\![x := a]\!](\sigma)$

$[\![x := \&y]\!](\sigma)$

$[\![x := *y]\!](\sigma)$

$[\![x := y]\!](\sigma)$

$[\![*x := y]\!](\sigma)$

```
/*  ∅ */ 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;
```

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
  - More intuition will be given in the sequel