Introduction to Abstract Interpretation

Armando Solar-Lezama
Computer Science and Artificial Intelligence Laboratory
MIT

With some slides from Saman Amarasinghe. Used with permission.

November 4, 2015
Course Recap

What you have learned so far

Operational Semantics
• How will a given program behave on a given input?
• This is the ground truth for any analysis

Types
• Annotations describe properties of the data that can be referred by a variable.
• Easy to describe properties that are global to the execution, but only one variable at a time (at least with the machinery we have seen here)
• Properties are fixed a priori by the type system designer
• Actual analysis is cheap
• Annotations can often be inferred

Program Logics
• Annotations describe properties of the state at a given point in the program.
• Easy to describe complex properties of the overall program state, but messy to describe properties that hold over time
• Logic provides a rich language for properties
• Actual analysis can be expensive
• Annotations are hard to infer
Some motivation

What is the loop invariant?

Intuition:
- The loop invariant is a set of states
- C transforms elements in $A \land b$ to other elements in $A$.

```plaintext
{true}
y=0;
while(x<10){
    x = x+1;
    y = y+2;
}
{even(y)}
```

$$\vdash \{A \land b\}c \{A\}$$

$$\vdash \{A\} while \ b \ do \ c \ \{A \land \lnot \ b\}$$
Simplifying the problem

This rule is strictly weaker
- Many correct programs can’t be proved with it

Simpler Intuition:
- The loop invariant is a set of states
- $C$ transforms elements in $A$ to other elements in $A.$
Discovering the invariant

There may be many candidates for $A$
- True is always an invariant

A2

Postcondition

A1

Precondition

Big $\iff$ Weak

A0
Discovering the invariant

We want a set $A$ such that $\neg \{A\}c \{A\}$
- It should be small enough to prove the postcondition (strong)
- But big enough to prove the precondition (weak)

Let $F(P) = wpc(c, P) \land Post$
- Then what we want is a greatest fixpoint solution of $A=F(A)$

Convergence properties
- Can we always find such solutions?

Forward vs. Backward
- When is it better to use $wpc$ vs. $spc$?

Precision
- How do we minimize the loss of precision?
Partial Orders

Set P

Partial order $\leq$ such that $\forall x, y, z \in P$

- $x \leq x$ (reflexive)
- $x \leq y$ and $y \leq x$ implies $x = y$ (asymmetric)
- $x \leq y$ and $y \leq z$ implies $x \leq z$ (transitive)

Can use partial order to define

- Upper and lower bounds
- Least upper bound
- Greatest lower bound
Upper Bounds

If $S \subseteq P$ then

- $x \in P$ is an upper bound of $S$ if $\forall y \in S. \ y \leq x$
- $x \in P$ is the least upper bound of $S$ if
  - $x$ is an upper bound of $S$, and
  - $x \leq y$ for all upper bounds $y$ of $S$
- $\lor$ - join, least upper bound, lub, supremum, sup
  - $\lor$ $S$ is the least upper bound of $S$
  - $x \lor y$ is the least upper bound of $\{x, y\}$
- Often written as $\sqcup$ as well
Lower Bounds

If $S \subseteq P$ then

- $x \in P$ is a lower bound of $S$ if $\forall y \in S. \ x \leq y$
- $x \in P$ is the greatest lower bound of $S$ if
  - $x$ is a lower bound of $S$, and
  - $y \leq x$ for all lower bounds $y$ of $S$

- $\wedge$ - meet, greatest lower bound, glb, infimum, inf
  - $\wedge S$ is the greatest lower bound of $S$
  - $x \wedge y$ is the greatest lower bound of $\{x, y\}$
- Often written as $\cap$ as well
Covering

\( x < y \) if \( x \leq y \) and \( x \neq y \)

\( x \) is covered by \( y \) (\( y \) covers \( x \)) if

- \( x < y \), and
- \( x \leq z < y \) implies \( x = z \)

Conceptually,

- \( y \) covers \( x \) if there are no elements between \( x \) and \( y \)
Lattices

If \( x \land y \) and \( x \lor y \) exist for all \( x,y \in P \)
then \( P \) is a **lattice**

If \( \land S \) and \( \lor S \) exist for all \( S \subseteq P \)
then \( P \) is a **complete lattice**

All finite lattices are complete

Example of a lattice that is not complete

- Integers \( I \)
- For any \( x, y \in I \), \( x \lor y = \max(x,y) \), \( x \land y = \min(x,y) \)
- But \( \lor I \) and \( \land I \) do not exist
- \( I \cup \{+\infty,-\infty\} \) is a complete lattice
Example

\[ P = \{ 000, 001, 010, 011, 100, 101, 110, 111 \} \]

(standard boolean lattice, also called hypercube)

\[ x \leq y \text{ if } (x \text{ bitwise and } y) = x \]

Hasse Diagram

- If \( y \) covers \( x \)
- Line from \( y \) to \( x \)
- \( y \) above \( x \) in diagram
Top and Bottom

Greatest element of $P$ (if it exists) is top ($\top$)
Least element of $P$ (if it exists) is bottom ($\bot$)
Connection Between $\leq$, $\land$, and $\lor$

The following 3 properties are equivalent:

- $x \leq y$
- $x \lor y = y$
- $x \land y = x$
Chains

A set $S$ is a chain if $\forall x, y \in S. \ y \leq x$ or $x \leq y$

P has no infinite chains if every chain in P is finite
Product Latices

Given two latices $L$ and $Q$, the product can easily be made a lattice

$$(l_1, q_1) \sqsubseteq (l_2, q_2) \iff l_1 \sqsubseteq l_2 \text{ and } q_1 \sqsubseteq q_2$$

For vectors of $L$, defining a lattice is also easy

$$\langle l_1, l_2, \ldots, l_k \rangle \sqsubseteq \langle t_1, t_2, \ldots, t_k \rangle \iff \forall i \in [1, k] \ l_i \sqsubseteq t_i$$
Back to our problem

A lattice of predicates:
- \(\langle x = \bot, even, odd, \top \rangle\)
  - Ex: \(\langle x = even, y = odd \rangle \sqsubseteq \langle x = \top, y = odd \rangle\)

What does this have to do with our problem?
Lattices and fixpoints

Order Preserving (Monotonic) Function:

\[ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \]

Now, let \( x_\bot \) be the least fixed point of \( f:L \rightarrow L \)
- so \( f(x_\bot) = x_\bot \)

Now, let \( x_0 = \bot \) and \( x_i = f(x_{i-1}) \)
- By induction, \( x_i \sqsubseteq x_\bot \)
- Also, the chain \( x_i \) is an ascending chain
- If \( L \) has no infinite ascending chains, sooner or later \( x_i = x_{i+1} = x_\bot \)

Same trick works for greatest fixed point!
- But then you have to start with \( x_0 = T \)
Back to our problem

\[
x = \begin{cases} \top & \text{Could be odd or even} \\ odd & \text{definitely odd} \\ even & \text{definitely even} \\ \bot & \text{who cares} \end{cases}
\]

A lattice of predicates:
- \( \langle x = \bot, even, odd, \top \rangle \)
  - Ex: \( \langle x = even, y = odd \rangle \subseteq \langle x = \top, y = odd \rangle \)

We now have a recipe to find a greatest fixpoint solution
- As long as \( F(P) = \text{wpc}(c, P) \wedge Post \) is monotonic in our latice
Finding a fixpoint

\[ \{x = \top, y = \top\} \]
\[
y = 0;
\]
\[
\text{while}(x < 10)\{
\quad x = x + 1;
\quad y = y + 2;
\}
\]
\[
\{x = \top, y = \text{even}\}
\]

Could be odd or even

definitely odd

definitely even

who cares

\[ F(P) = \text{wpc}(c, P) \land Post \]
- \[ P_0 = \{x = \top, y = \top\} \]
- \[ P_1 = \{x = \top, y = \text{even}\} \]
- \[ P_2 = \{x = \top, y = \text{even}\} \]
- Success!
Complicating things a bit

\{x = T, y = T\}

\text{y=0; t=1; } \quad \text{c0}

\text{while(x<10){}
  \text{x = x+1; } \quad \text{c1}
  \text{y = y+2; } \quad \text{c2}
  \text{if(x=5){
    \text{t=t+2; } \quad \text{c2}
  \text{else{
    \text{y = t+1; } \quad \text{c3}
  }\}
}\}

\{x = T, y = even\}

\text{)}

\begin{align*}
\vdash & \{A \land b\} c_1 \{B\} \quad \vdash \{A \land \text{not } b\} c_2 \{B\} \\
\vdash & \{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}
\end{align*}

\text{Relaxed Rule}

\begin{align*}
\vdash & \{A \land b\} c_1 \{B\} \quad \vdash \{A \land \text{not } b\} c_2 \{B\} \\
\vdash & \{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}
\end{align*}

\text{F(P) = wpc(c, P) \land Post} \\
\quad = wpc(c_1, wpc(c_2, P) \land wpc(c_3, P)) \land \text{Post}
**Dataflow equations**

\[
\begin{align*}
\{x = T, y = T\} & \quad <-P1 \\
y = 0; t = 1; & \quad <-C0 \\
\text{while}(x < 10)\{ & \quad <-P2 \\
\quad x = x+1; & \quad <-C1 \\
\quad y = y+2; & \quad <-C1 \\
\quad \text{if}(x = 5)\{ & \quad <-P3 \\
\quad \quad t = t+2; & \quad <-C2 \\
\quad \} & \quad <-P3 \\
\quad \} & \quad <-P2 \\
\{x = T, y = even\} & \quad <-P5
\end{align*}
\]

\[
F(P) = wpc(c, P) \land Post = wpc(c1, wpc(c2, P) \land wpc(c3, P)) \land Post
\]

\[
p1 \sqsubseteq wpc(c0, p2) \\
p2 \sqsubseteq wpc(c1, p3) \\
p3 \sqsubseteq wpc(c2, p2) \land wpc(c3, p2) \\
p2 \sqsubseteq p5 \\
p2 \sqsubseteq wpc(c1, p3) \land p5
\]

Big <=> Weak
So \( A \Rightarrow B \) is equivalent to \( A \sqsubseteq B \)
Dataflow equations

\{x = T, y = T\} \leftarrow P1

y = 0; t = 1; \quad \{c_0\}

while (x < 10) { \leftarrow P2
  x = x + 1; \quad \{c_1\}
  y = y + 2; \quad \{c_1\}
  if (x = 5) {
    t = t + 2; \quad \{c_2\}
  } else {
    y = t + 1; \quad \{c_3\}
  }
}\leftarrow P2

\{x = T, y = even\} \leftarrow P5

p1 \equiv wpc(c_0, p2)

p2 \equiv wpc(c_1, p3) \land p5

p3 \equiv wpc(c_2, p2) \land wpc(c_3, p2)
Dataflow Analysis

General Analysis Framework
- Developed by Kildall in 1973
- Traditionally used for compiler optimization

Frame analysis question as a set of equations on a CFG
\{x = T, y = T\} \quad \text{<-P1}

y=0; \ t=1;

while(x<10){ \quad \text{<-P2}
    x = x+1;
    y = y+2;
    if(x=5){ \quad \text{<-P3}
        t=t+2;
    } \text{else}{
        y = t+1;
    }
} \quad \text{<-P2}

\{x = T, y = \text{even}\} \quad \text{<-P5}
Control Flow Graph

Very general program representation
- Easy to represent unstructured control flow
- Widely used by most program analysis tools for imperative languages
Solution strategy

For every basic block we have an equation of the form

- $Out \subseteq F(in)$
- Use meet ($\land$) when many edges meet together

We can solve through “Chaotic Iteration”

- Keep a list of nodes to update
- Pick one CFG node at a time
- Update $out$ from new $in$
- If out changed, add its children to the list

```
y=0;
t=1;
x=x+1;
y=y+2;
t=t+2;
y=t-1;
```
Computing transfer function

So far we defined it in terms of weakest precondition.
- Or alternatively, strongest postcondition
- Too general and expensive!

We can hard-code a transfer function specific to the lattice
- For finite lattices they can be implemented cheaply in terms of bitvector operations

We can build lattices for arbitrary facts about the program
- Need to make sure our transfer functions are monotonic
Example: Reaching Definitions

Concept of definition and use
- $a = x + y$
- is a definition of $a$
- is a use of $x$ and $y$

A definition reaches a use if
- value written by definition
- may be read by use
Reaching Definitions

\[
\begin{align*}
  s &= 0; \\
  a &= 4; \\
  i &= 0; \\
  k &= 0 \\
  b &= 1; \\
  b &= 2; \\
  i &< n \\
  s &= s + a \times b; \\
  i &= i + 1; \\
  \text{return } s
\end{align*}
\]
Reaching Definitions and Constant Propagation

Is a use of a variable a constant?
- Check all reaching definitions
- If all assign variable to same constant
- Then use is in fact a constant

Can replace variable with constant
Is a Constant in \( s = s + a \cdot b \)?

Yes!

On all reaching definitions
\( a = 4 \)
Constant Propagation Transform

```plaintext
s = 0;
a = 4;
i = 0;
k == 0

b = 1;
b = 2;
i < n
s = s + 4*b;
i = i + 1;
return s
```

Yes!
On all reaching definitions
a = 4
Is b Constant in $s = s + a \times b$?

No!
One reaching definition with $b = 1$
One reaching definition with $b = 2$
Computing Reaching Definitions

Compute with sets of definitions
- represent sets using bit vectors
- each definition has a position in bit vector

At each basic block, compute
- definitions that reach start of block
- definitions that reach end of block

Do computation by simulating execution of program until reach fixed point
1:  s = 0;
2:  a = 4;
3:  i = 0;
   k == 0
4:  b = 1;
5:  b = 2;
6:  s = s + a*b;
   7:  i = i + 1;
    return s
Transfer functions

Each basic block has
- **IN** - set of definitions that reach beginning of block
- **OUT** - set of definitions that reach end of block
- **GEN** - set of definitions generated in block
- **KILL** - set of definitions killed in block

**GEN**
\[
\begin{align*}
s &= s + a \times b; \\ i &= i + 1;
\end{align*}
\] = 0000011

**KILL**
\[
\begin{align*}
s &= s + a \times b; \\ i &= i + 1;
\end{align*}
\] = 1010000

Analyzer scans each basic block to derive **GEN** and **KILL** sets for each function
Dataflow Equations

\[ \text{IN}[b] = \text{OUT}[b_1] \cup \ldots \cup \text{OUT}[b_n] \]
- where \( b_1, \ldots, b_n \) are predecessors of \( b \) in CFG

\[ \text{OUT}[b] = (\text{IN}[b] - \text{KILL}[b]) \cup \text{GEN}[b] \]

\[ \text{IN}[\text{entry}] = 0000000 \]

Result: system of equations
Solving Equations

Use fixed point algorithm
Initialize with solution of $\text{OUT}[b] = 0000000$
Repeatedly apply equations
- $\text{IN}[b] = \text{OUT}[b1] \cup \ldots \cup \text{OUT}[bn]$
- $\text{OUT}[b] = (\text{IN}[b] - \text{KILL}[b]) \cup \text{GEN}[b]$
Until reach fixed point
Until equation application has no further effect
Use a worklist to track which equation applications may have a further effect
Questions

Does the algorithm halt?
- yes, because transfer function is monotonic
- if increase IN, increase OUT
- in limit, all bits are 1

If bit is 0, does the corresponding definition ever reach basic block?

If bit is 1, is does the corresponding definition always reach the basic block?