Program Analysis: Wrapping Up

Uday Khedker
(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

November 2017
Part 1

About These Slides
Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

  
  (Indian edition published by Ane Books in 2013)

Apart from the above book, some slides are based on the material from the following books


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.*
Part 2

The Big Picture
So what have learnt?

*Education is what remains after you have forgotten everything that was taught*

- Albert Einstein
The Main Theme of the Course

Constructing suitable abstractions for sound & precise modelling of runtime behaviour of programs efficiently
The Main Theme of the Course

Constructing *suitable abstractions* for
*sound & precise modelling* of
*runtime behaviour* of programs
*efficiently*

![Diagram showing static and dynamic program code]

Static

Program Code

Dynamic
The Main Theme of the Course

Constructing suitable abstractions for sound & precise modelling of runtime behaviour of programs efficiently

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Dynamic

Program Execution

Nov 2017
The Main Theme of the Course

Constructing suitable abstractions for sound & precise modelling of runtime behaviour of programs efficiently

Abstract, Bounded, Single Instance

Static
- Program Code

Concrete, Unbounded, Infinitely Many

Dynamic
- Program Execution
- Memory
The Main Theme of the Course

Constructing suitable abstractions for sound & precise modelling of runtime behaviour of programs efficiently.

Abstract, Bounded, Single Instance

Static

Program Code

Summary Information

Concrete, Unbounded, Infinitely Many

Dynamic

Program Execution

Memory
The Main Theme of the Course

Constructing *suitable abstractions* for
*sound & precise modelling* of
*runtime behaviour* of programs
*efficiently*

---

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

**Static**

- Program Code
- Static Analysis
- Summary Information

**Dynamic**

- Program Execution
- Memory
- Memory
### Soundness and Precision of Static Analysis

#### Example Program

```c
int a;
int f(int b)
{
    int c;
    c = a*2;
    while (b <= c)
        b = b+1;
    if (b < 9)
        b = b+a;
    return b;
}
```

#### Simplified IR

<table>
<thead>
<tr>
<th>Simplified IR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

#### Control Flow Graph

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
## Soundness and Precision of Static Analysis

### Example Program

```c
int a;
int f(int b)
{
  int c;
  c = a*2;
  while (b <= c)
  {
    b = b+1;
  }
  if (b < 9)
  {
    b = b+a;
  }
  return b;
}
```

### Simplified IR

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c = a*2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>if (b &gt; c) goto 5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b = b + 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>goto 2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>if (b ≥ 9) goto 7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b = b+a</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>return b</td>
<td></td>
</tr>
</tbody>
</table>

### Control Flow Graph
Soundness and Precision of Static Analysis

Example Program

```c
int a;
int f(int b)
{
    int c;
    c = a*2;
    while (b <= c)
    {
        b = b+1;
    }
    if (b < 9)
    {
        b = b+a;
    }
    return b;
}
```

Simplified IR

1: c = a*2
2: if (b > c) goto 5
3: b = b + 1
4: goto 2
5: if (b ≥ 9) goto 7
6: b = b+a
7: return b

Control Flow Graph

```
c = a*2

if (b > c)
    F

if (b ≥ 9)
    F

b = b + 1

b = b+a

return b
```
Execution Traces for Concrete Semantics

- A state: (Program Point, Variables $\mapsto$ Values)
- A trace: a valid sequence of states starting with a given initial state

1: $c = a \times 2$
2: if ($b > c$)
   goto 5
3: $b = b + 1$
4: goto 2
5: if ($b \geq 9$)
   goto 7
6: $b = b + a$
7: return $b$
Execution Traces for Concrete Semantics

- A state: (Program Point, Variables \( \mapsto \) Values)
- A trace: a valid sequence of states starting with a given initial state

\[
\begin{align*}
\text{Trace 1} \\
&\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
0 : & (1,2,3) \\
1 : & (1,2,2) \\
2 : & (1,2,2) \\
3 : & (1,3,2) \\
4 : & (1,3,2) \\
5 : & (1,3,2) \\
6 : & (1,4,2) \\
7 : & (1,4,2)
\end{array}
\end{align*}
\]
**Execution Traces for Concrete Semantics**

- A state: (Program Point, Variables $\mapsto$ Values)
- A trace: a valid sequence of states starting with a given initial state

<table>
<thead>
<tr>
<th>Trace 1</th>
<th>Trace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a</strong></td>
<td><strong>a</strong></td>
</tr>
<tr>
<td><strong>b</strong></td>
<td><strong>b</strong></td>
</tr>
<tr>
<td><strong>c</strong></td>
<td><strong>c</strong></td>
</tr>
<tr>
<td><strong>0</strong> : (1, 2, 3)</td>
<td><strong>0</strong> : (5, 10, 7)</td>
</tr>
<tr>
<td><strong>1</strong> : (1, 2, 2)</td>
<td><strong>1</strong> : (5, 10, 10)</td>
</tr>
<tr>
<td><strong>2</strong> : (1, 2, 2)</td>
<td><strong>2</strong> : (5, 10, 10)</td>
</tr>
<tr>
<td><strong>3</strong> : (1, 3, 2)</td>
<td><strong>3</strong> : (5, 11, 10)</td>
</tr>
<tr>
<td><strong>4</strong> : (1, 3, 2)</td>
<td><strong>4</strong> : (5, 11, 10)</td>
</tr>
<tr>
<td><strong>5</strong> : (1, 4, 2)</td>
<td><strong>5</strong> : (5, 11, 10)</td>
</tr>
<tr>
<td><strong>6</strong> : (1, 4, 2)</td>
<td><strong>7</strong> : (5, 11, 10)</td>
</tr>
</tbody>
</table>
Execution Traces for Concrete Semantics

- A state: (Program Point, Variables \(\mapsto\) Values)
- A trace: a valid sequence of states starting with a given initial state

```
1: c = a*2
2: if (b > c) goto 5
3: b = b + 1
4: goto 2
5: if (b \geq 9) goto 7
6: b = b+a
7: return b
```

- Number of traces is potentially infinite

Trace 1
```
a b c
0 : (1, 2, 3)
1 : (1, 2, 2)
2 : (1, 2, 2)
3 : (1, 3, 2)
4 : (1, 3, 2)
5 : (1, 3, 2)
6 : (1, 4, 2)
```

Trace 2
```
a b c
0 : (5, 10, 7)
1 : (5, 10, 10)
2 : (5, 10, 10)
3 : (5, 11, 10)
4 : (5, 11, 10)
5 : (5, 11, 10)
6 : (5, 11, 10)
7 : (5, 11, 10)
```

Nov 2017
**Execution Traces for Concrete Semantics**

- A state: (Program Point, Variables $\mapsto$ Values)
- A trace: a valid sequence of states starting with a given initial state

<table>
<thead>
<tr>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$, $b$, $c$</td>
<td>$a$, $b$, $c$</td>
<td>$a$, $b$, $c$</td>
</tr>
<tr>
<td>1: $c = a\times2$</td>
<td>1: $c = a\times2$</td>
<td>0: $(-1, 1, 6)$</td>
</tr>
<tr>
<td>2: if ($b &gt; c$) goto 5</td>
<td>2: if ($b &gt; c$) goto 5</td>
<td>1: $(-1, 1, -2)$</td>
</tr>
<tr>
<td>3: $b = b + 1$</td>
<td>3: $b = b + 1$</td>
<td>2: $(-1, 1, -2)$</td>
</tr>
<tr>
<td>4: goto 2</td>
<td>4: goto 2</td>
<td>3: $(-1, 2, -2)$</td>
</tr>
<tr>
<td>5: if ($b \geq 9$) goto 7</td>
<td>5: if ($b \geq 9$) goto 7</td>
<td>4: $(-1, 2, -2)$</td>
</tr>
<tr>
<td>6: $b = b+a$</td>
<td>6: $b = b+a$</td>
<td>2: $(-1, 3, -2)$</td>
</tr>
<tr>
<td>7: return $b$</td>
<td>7: return $b$</td>
<td>3: $(-1, 3, -2)$</td>
</tr>
</tbody>
</table>

- Number of traces is potentially infinite
- Not all traces may terminate

7: $(1, 4, 2)$
Execution Traces for Concrete Semantics

- A state: (Program Point, Variables $\mapsto$ Values)
- A trace: a valid sequence of states starting with a given initial state

```
1: c = a*2
2: if (b > c) goto 5
3: b = b + 1
4: goto 2
5: if (b $\geq$ 9) goto 7
6: b = b + a
7: return b
```

<table>
<thead>
<tr>
<th>Trace 1</th>
<th>Trace 2</th>
<th>Trace 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c a b c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0: (1, 2, 3)</td>
<td>0: (−1, 1, 6)</td>
<td></td>
</tr>
<tr>
<td>1: (1, 2, 2)</td>
<td>1: (−1, 1, −2)</td>
<td></td>
</tr>
<tr>
<td>2: (1, 2, 2)</td>
<td>2: (−1, 1, −2)</td>
<td></td>
</tr>
<tr>
<td>3: (1, 2, 2)</td>
<td>3: (−1, 2, −2)</td>
<td></td>
</tr>
<tr>
<td>4: (1, 2, 2)</td>
<td>4: (−1, 2, −2)</td>
<td></td>
</tr>
<tr>
<td>5: (1, 2, 2)</td>
<td>5: (−1, 2, −2)</td>
<td></td>
</tr>
<tr>
<td>6: (1, 2, 2)</td>
<td>6: (−1, 3, −2)</td>
<td></td>
</tr>
<tr>
<td>7: (1, 4, 2)</td>
<td>7: (−1, 3, −2)</td>
<td></td>
</tr>
</tbody>
</table>

- Number of traces is potentially infinite
- Not all traces may terminate
- We consider only terminating traces
Static Analysis Computes Abstractions of Traces

Traces

Execution Time
Static Analysis Computes Abstractions of Traces
Static Analysis Computes Abstractions of Traces
Static Analysis Computes Abstractions of Traces

For compile time modelling of possible runtime behaviours of a program

- compute a set of states that cover all traces
- associate the sets with appropriate program points

States may be defined in terms of properties derived from values of variables
An over-approximation of traces is sound
Soundness of Abstractions

Sound

- An over-approximation of traces is sound

Unsound

- Missing any state in any trace causes unsoundness
Precision of Sound Abstractions

Imprecise
Precision of Sound Abstractions

Imprecise

More Precise
Precision of Sound Abstractions

Imprecise

More Precise

Even More Precise
Precision of Sound Abstractions

- Precision is relative, soundness is absolute
- Qualifiers “more precise” and “less precise” are meaningful
- Qualifiers “more sound” and “less sound” are not meaningful
Motifs Used for Building the Theme
Motifs Used for Building the Theme

- Intuition-formalism dichotomy

- Intuitions representing abstract view of the run time behaviour

- Systematic formulation amenable to automation and reasoning
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy

- Separate reasoning from the implementation
- Systematize construction of analyzers
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy

- Formalizing underlying concepts rigorously
- Formulating analysis in terms of data flow equations (confluence, initialization, boundary info, flow functions etc.)
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations

- Generalize by relaxing conditions
  (Previous abstractions should become special cases)
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations

- Generalize by relaxing conditions (Previous abstractions should become special cases)
- Generalize the intuitions, specifications, or algorithm
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations

• Generalize by relaxing conditions
  (Previous abstractions should become special cases)
• Generalize the intuitions, specifications, or algorithm
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations

- Generalize by relaxing conditions
  (Previous abstractions should become special cases)
- Generalize the intuitions, specifications, or algorithm
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations
- Filtering and distilling ideas

- Ask the right questions
- Separate relevant from irrelevant
Motifs Used for Building the Theme

- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations
- Filtering and distilling ideas
- Working from first principles

**First principles:** A small set of orthogonal concepts

- Add as few concepts as possible to the set of first principles
Seeking Generalizations

Intuitions about program behaviour

Formalization/Formulation

Algorithm
Module 1: Bit Vector Frameworks

Intuitions

Data flow information at a program point $u$
- represents information valid for all execution instances of $u$
- depends on some or all paths,
- starting at, or ending at, or passing through $u$
- may be generated, killed, or propagated

Formalization

Algorithm
Module 1: Bit Vector Frameworks

Intuitions

Data flow information at a program point $u$
- represents information valid for all execution instances of $u$
- depends on some or all paths,
- starting at, or ending at, or passing through $u$
- may be generated, killed, or propagated

Representations

- programs $\equiv$ control flow graphs
- data flow values $\equiv$ sets or bit vectors
- dependence of data flow values $\equiv$ data flow equations

Formalization

Algorithm
Module 1: Bit Vector Frameworks

**Intuitions**

Data flow information at a program point $u$
- represents information valid for all execution instances of $u$
- depends on some or all paths,
- starting at, or ending at, or passing through $u$
- may be generated, killed, or propagated

**Formalization**

Representations
- programs $\equiv$ control flow graphs
- data flow values $\equiv$ sets or bit vectors
- dependence of data flow values $\equiv$ data flow equations

**Algorithm**

- convergence
- iterative refinement
- initialization
- round robin method
Module 2: Theoretical Abstractions

- sound approximation of data flow information
- merging data flow values
- direction of flow, relationship with graph traversal
- desired vs. computable solution
Module 2: Theoretical Abstractions

Intuitions

- sound approximation of data flow information
- merging data flow values
- direction of flow, relationship with graph traversal
- desired vs. computable solution

Formalization

- lattices, partial order, meet, descending chain condition (DCC)
- monotonicity, distributivity and non-separability of flow functions
- MFP and MoP assignments
- information flow paths, depth and width of a CFG

Algorithm

Nov 2017
Module 2: Theoretical Abstractions

Intuitions
- sound approximation of data flow information
- merging data flow values
- direction of flow, relationship with graph traversal
- desired vs. computable solution

Formalization
- lattices, partial order, meet, descending chain condition (DCC)
- monotonicity, distributivity and non-separability of flow functions
- MFP and MoP assignments
- information flow paths, depth and width of a CFG

Algorithm
- conservative initialization
- complexity
- work list based method
Module 2: Theoretical Abstractions

- sound approximation of data flow information
- merging data flow values
- direction of flow, relationship with graph traversal
- desired vs. computable solution

Intuitions

- lattices
- monotonicity
- MFP and MoP assignments
- information flow paths, depth and width of a CFG
- conservative initialization
- complexity
- work list based method

Formalization

- Theme: Generalization in formulations
- Learning outcome: Add the following requirements to the set of first principles
  Monotonic flow functions and meet semi-lattice satisfying DCC
Module 3: General Frameworks

- dependence of data flow values across entities
- generation and killing depending upon the incoming information
- flow insensitivity, may and must nature in flow sensitivity
- use of program point in data flow information
Module 3: General Frameworks

Intuitions
- dependence of data flow values across entities
- generation and killing depending upon the incoming information
- flow insensitivity, may and must nature in flow sensitivity
- use of program point in data flow information

Formalization
- Representations for data flow values: Sets, tuples, strings, graphs
- modelling non-separability in flow functions using dependent parts
- flow function operations
  (e.g. path removal, factorization, extension, relation application)

Algorithm

Nov 2017
Module 3: General Frameworks

- dependence of data flow values across entities
- generation and killing depending upon the incoming information
- flow insensitivity, may and must nature in flow sensitivity
- use of program point in data flow information

Intuitions

- Generalizations in formulation

Observations:

Structure of heap accesses consist of repeating patterns that resemble the program structure

Program analysis should be driven by liveness to restrict the information to usable information
Module 4: Interprocedural Data Flow Analysis

- interprocedural validity of paths and context sensitivity
- constructing summary flow functions Vs. propagating data flow values
- orthogonality of context and data flow information
- partitioning contexts based on data flow values
Module 4: Interprocedural Data Flow Analysis

Intuitions

- interprocedural validity of paths and context sensitivity
- constructing summary flow functions Vs. propagating data flow values
- orthogonality of context and data flow information
- partitioning contexts based on data flow values

Formalization

- lattices of flow functions, reducing function compositions and meets
- data flow equations for constructing summary flow functions
- value contexts, their exit values, and transitions
Module 4: Interprocedural Data Flow Analysis

- interprocedural validity of paths and context sensitivity
- constructing summary flow functions Vs. propagating data flow values
- orthogonality of context and data flow information
- partitioning contexts based on data flow values

Intuitions

Formalization

- lattices of flow functions, reducing function compositions and meets
- data flow equations for constructing summary flow functions
- value contexts, their exit values, and transitions

Algorithm

- work list based method
  ordering of nodes in post or reverse post order
Module 4: Interprocedural Data Flow Analysis

- interprocedural validity of paths and context sensitivity
- constructing summary flow functions Vs. propagating data flow values
- orthogonality of context and data flow information
- partitioning contexts based on data flow values

Intuitions

- lattice structures
- data flow values
- value contexts, their exit values, and transitions

Formalization

- Generalizations in formulation and algorithm
- Observation:
  Separating relevant information from irrelevant information can have a significant impact

Algorithm

- work list based method
  ordering of nodes in post or reverse post order
Sequence of Generalizations in the Course Modules
Sequence of Generalizations in the Course Modules

Bit Vector Frameworks

Theoretical abstractions
Sequence of Generalizations in the Course Modules

- Theoretical abstractions
- General frameworks
- Bit Vector Frameworks
Sequence of Generalizations in the Course Modules

- Theoretical abstractions
- General frameworks
- Bit Vector Frameworks
- Intraprocedural Level
Sequence of Generalizations in the Course Modules

Intraprocedural Level

Interprocedural Level

Bit Vector Frameworks

General frameworks

Theoretical abstractions
Takeaways of the Course

• Data Flow Analysis:

  Minimal conditions for devising a data flow framework

  ▶ Intraprocedural formulation:
    - Meet semilattice satisfying the descending chain condition, and
    - Monotonic flow functions

  ▶ Extension to interprocedural level: Additional restrictions
    - Value based approach: Finiteness of lattice
    - Functional approach: Distributive primitive entity functions
Takeaways of the Course

• Data Flow Analysis:

  Minimal conditions for devising a data flow framework
  
  ▶ Intraprocedural formulation:
    – Meet semilattice satisfying the descending chain condition, and
    – Monotonic flow functions
  
  ▶ Extension to interprocedural level: Additional restrictions
    – Value based approach: Finiteness of lattice
    – Functional approach: Distributive primitive entity functions

• General:

  ▶ Generalization, refinements, distilling the essence
  ▶ Asking the right questions
  ▶ Separating relevant information from the irrelevant information
Still Bigger Picture . . .

Scope of the course: Generic static analyses for imperative languages

Did not cover
Still Bigger Picture . . .

Scope of the course: Generic static analyses for imperative languages

Did not cover

- Influences of other languages features
  - Concurrency, Object orientation, Coroutines, Exception handling
  - Declarative paradigms: functional or logic languages
Still Bigger Picture . . .

Scope of the course: Generic static analyses for imperative languages

Did not cover

- Influences of other languages features
  - Concurrency, Object orientation, Coroutines, Exception handling
  - Declarative paradigms: functional or logic languages

- Influences of other goals
  - Verification and validation, testing (e.g. analyses for finding bugs does not require exhaustiveness or soundness)
  - Path sensitive analyses
  - Shape analysis
  - Optimization specific analyses
  - Adhoc techniques of achieving efficiency
  - Analyses for JIT compilation
  - Parallelization, Vectorization, Dependence analysis
Still Bigger Picture . . .

Scope of the course: Generic static analyses for imperative languages

Did not cover

- Influences of other languages features
  - Concurrency, Object orientation, Coroutines, Exception handling
  - Declarative paradigms: functional or logic languages

- Influences of other goals
  - Verification and validation, testing (e.g. analyses for finding bugs does not require exhaustiveness or soundness)
  - Path sensitive analyses
  - Shape analysis
  - Optimization specific analyses
  - Adhoc techniques of achieving efficiency
  - Analyses for JIT compilation
  - Parallelization, Vectorization, Dependence analysis

- Other analysis methods
  - Abstract interpretation, Type inference, Constraint resolution
Last But Not the Least

Thank You!