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

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.*
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

---

**Soundness and Precision of Static Analysis**

**Example Program**

```plaintext
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**

```plaintext
\[
\begin{align*}
\text{c = a*2} \\
\text{if (b>c)} \\
\text{\quad b = b+1} \\
\text{\quad T} \\
\text{if (b\geq9)} \\
\text{\quad F} \\
\text{\quad b = b+a} \\
\text{\quad T} \\
\text{\quad \quad b = b+a} \\
\text{\quad \quad \quad return b}
\end{align*}
\]
```

**Control Flow Graph**

**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

**Trace 1**

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

**Trace 2**

<table>
<thead>
<tr>
<th>States</th>
<th>Trace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c</td>
<td>0 : (5, 10, 7)</td>
</tr>
<tr>
<td>1: c = a*2</td>
<td>1 : (5, 10, 10)</td>
</tr>
<tr>
<td>2: if (b &gt; c) goto 5</td>
<td>2 : (5, 10, 10)</td>
</tr>
<tr>
<td>3: b = b + 1</td>
<td>3 : (5, 11, 10)</td>
</tr>
<tr>
<td>4: goto 2</td>
<td>4 : (5, 11, 10)</td>
</tr>
<tr>
<td>5: if (b \geq 9) goto 7</td>
<td>5 : (5, 11, 10)</td>
</tr>
<tr>
<td>6: b = b+a</td>
<td>5 : (5, 11, 10)</td>
</tr>
<tr>
<td>7: return b</td>
<td>7 : (5, 11, 10)</td>
</tr>
</tbody>
</table>
Execution Traces for Concrete Semantics

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

\[
\begin{align*}
    &1: c = a*2 \\
    &2: \text{if } (b > c) \quad \text{goto 5} \\
    &3: b = b + 1 \\
    &4: \text{goto 2} \\
    &5: \text{if } (b \geq 9) \quad \text{goto 7} \\
    &6: b = b+a \\
    &7: \text{return } b \\
\end{align*}
\]

\[
\begin{align*}
    &7: (1,4,2)
\end{align*}
\]

\[
\begin{align*}
    &1: c = a*2 \\
    &2: \text{if } (b > c) \quad \text{goto 5} \\
    &3: b = b + 1 \\
    &4: \text{goto 2} \\
    &5: \text{if } (b \geq 9) \quad \text{goto 7} \\
    &6: b = b+a \\
    &7: \text{return } b \\
\end{align*}
\]

\[
\begin{align*}
    &7: (1,4,2)
\end{align*}
\]
Static Analysis Computes Abstractions of Traces

Traces

An Abstraction 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

Soundness of Abstractions

Sound
- An over-approximation of traces is sound
- Missing any state in any trace causes unsoundness
Unsound

Precision of Sound Abstractions

Imprecise

More Precise

Even More Precise

- 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

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

- Intuitions representing abstract view of the run time behaviour
- Systematic formulation amenable to automation and reasoning
- Separate reasoning from the implementation
- Systematize construction of analyzers
- Generalize by relaxing conditions (Previous abstractions should become special cases)
- Generalize the intuitions, specifications, or algorithm

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
- Filtering and distilling ideas

- Ask the right questions
- Separate relevant from irrelevant

Module 1: Bit Vector Frameworks

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

- convergence
- iterative refinement
- initialization
- round robin method
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

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

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
- 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

- 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

- Algorithm
  - Work list based method
    - Ordering of nodes in post or reverse post order

Intuitions
Formalization
Algorithm

Sequence of Generalizations in the Course Modules

- Bit Vector
- Frameworks

Theoretical abstractions
Sequence of Generalizations in the Course Modules

Intraprocedural Level

Interprocedural Level

- 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

- 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

Thank You!