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:

- Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press (Taylor and Francis Group). 2009.

  (Indian edition published by Ane Books in 2013)

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

- A. V. Aho, M. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley. 2006.
- M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland Inc. 1977.

Part 2

The Big Picture

So what have learnt?

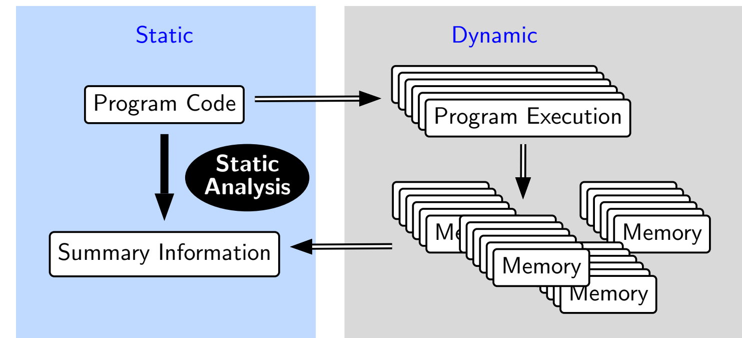
Education is what remains after you have forgotten everything that was taught

- Albert Einstein

The Main Theme of the Course

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

Abstract, Bounded, Single Instance — Concrete, Unbounded, Infinitely Many



Soundness and Precision of Static Analysis

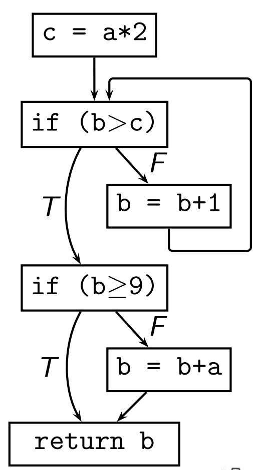
Example Program

```
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



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 ≥ 9)
       goto 7
6: b = b+a
7: return b
```

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

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 ≥ 9)
      goto 7
6: b = b+a
7: return b
```

Trace 1 *a b c*

Trace 2 *a b c*

- Number of traces is potentially infinite

$7:(1,4,2)$

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 ≥ 9)
      goto 7
6: b = b+a
7: return b
```

Trace 1 *a b c*

Trace 2 *a b c*

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

$7:(1,4,2)$

Trace 3

a b c

$0:(-1,1,6)$
$1:(-1,1,-2)$
$2:(-1,1,-2)$
$3:(-1,2,-2)$
$4:(-1,2,-2)$
$2:(-1,2,-2)$
$3:(-1,3,-2)$
$4:(-1,3,-2)$
$2:(-1,3,-2)$
$\ldots$

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 ≥ 9)
      goto 7
6: b = b+a
7: return b
```

Trace 1 *a b c*

Trace 2 *a b c*

- Number of traces is potentially infinite
- Not all traces may terminate
- We consider only terminating traces

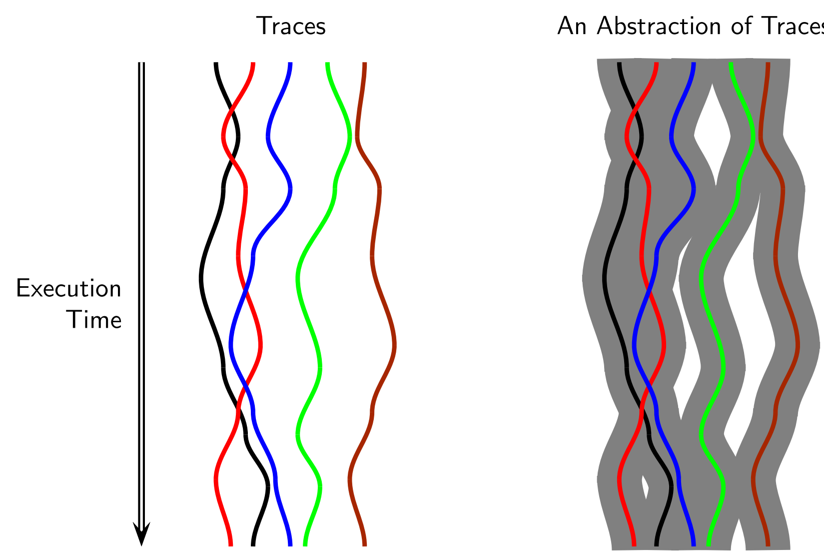
$7:(1,4,2)$

Trace 3

a b c

$0:(-1,1,6)$
$1:(-1,1,-2)$
$2:(-1,1,-2)$
$3:(-1,2,-2)$
$4:(-1,2,-2)$
$2:(-1,2,-2)$
$3:(-1,3,-2)$
$4:(-1,3,-2)$
$2:(-1,3,-2)$
$\ldots$

Static Analysis Computes Abstractions of Traces

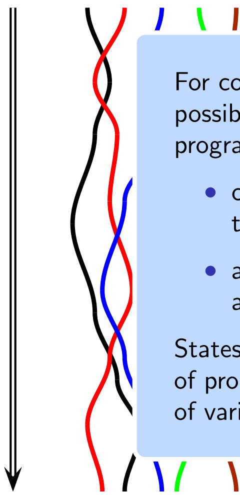


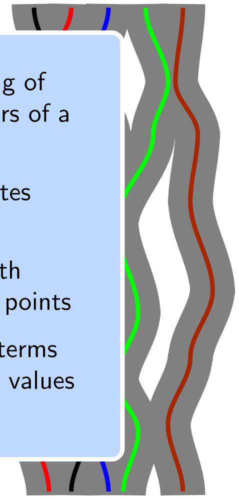
Static Analysis Computes Abstractions of Traces

Traces

An Abstraction of Traces

Execution Time





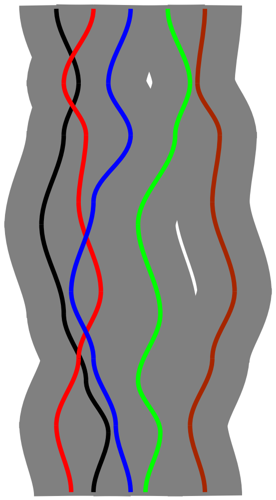
For compile time modelling of possible runtime behaviours of a program

- 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

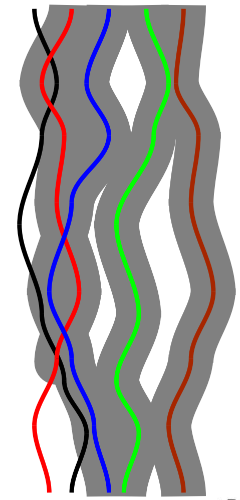
Soundness of Abstractions

Sound



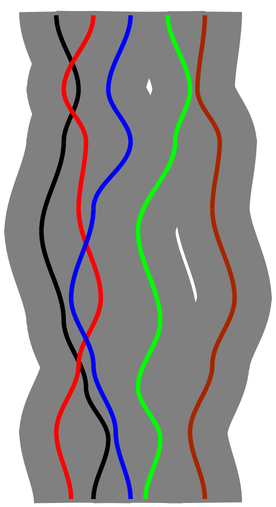
- An over-approximation of traces is sound
- Missing any state in any trace causes unsoundness

Unsound

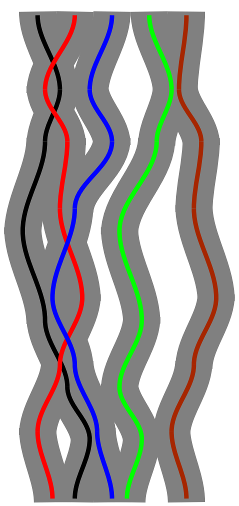


Precision of Sound Abstractions

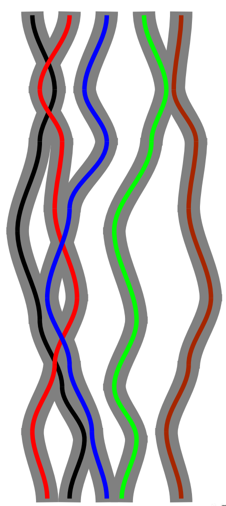
Imprecise



More Precise



Even More Precise

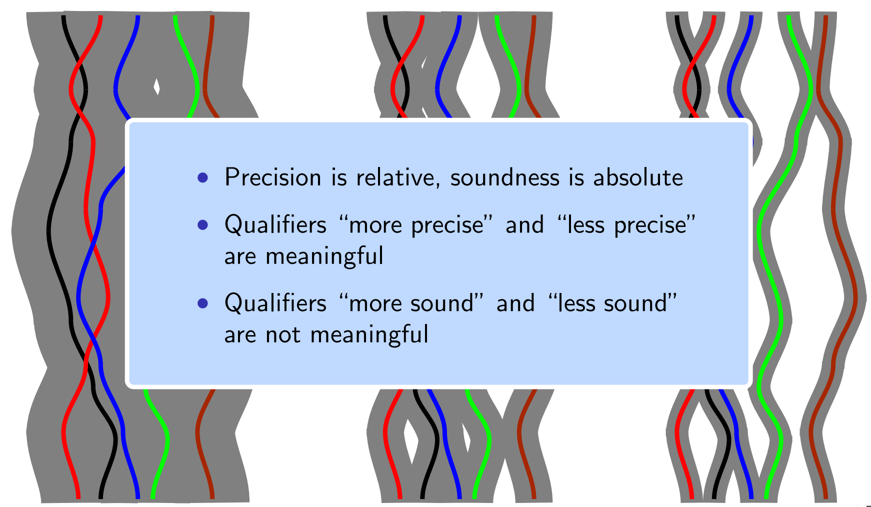


Precision of Sound Abstractions

Imprecise

More Precise

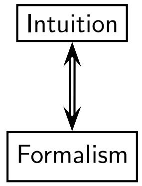
Even More Precise



- 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

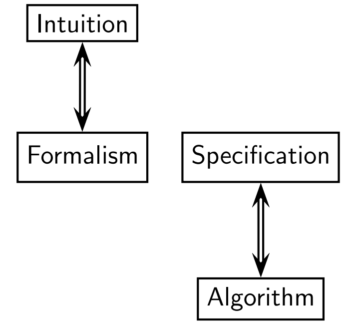
- Intuition-formalism dichotomy



- Intuitions representing abstract view of the run time behaviour
- Systematic formulation amenable to automation and reasoning

Motifs Used for Building the Theme

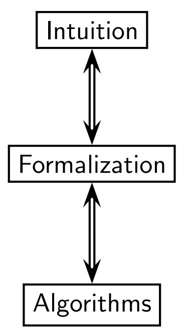
- Intuition-formalism dichotomy
- Specification-implementation dichotomy



- Separate reasoning from the implementation
- Systematize construction of analyzers

Motifs Used for Building the Theme

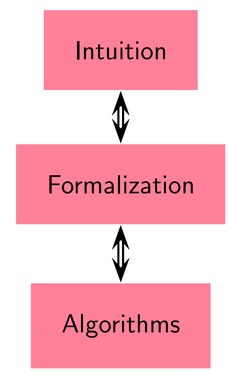
- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations



- 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

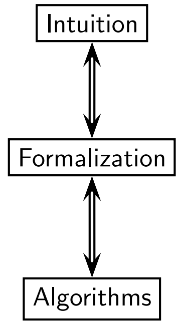
- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations



- Generalize by relaxing conditions
  (Previous abstractions should become special cases)
- Generalize the intuitions, specifications, or algorithm

Motifs Used for Building the Theme

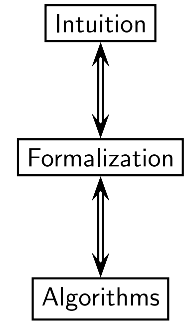
- Intuition-formalism dichotomy
- Specification-implementation dichotomy
- Successive generalizations
- Filtering and distilling ideas



- Ask the right questions
- Separate relevant from irrelevant

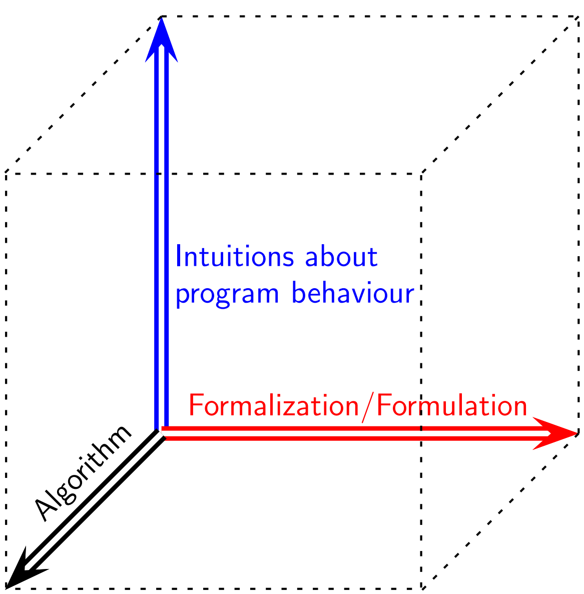
Motifs Used for Building the Theme

- 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



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 ≡ control flow graphs
— data flow values ≡ sets or bit vectors
— dependence of data flow values ≡ data flow equations

Algorithm

— 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

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
— monoto
— MFP a
— informa

- 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

Algorithm

— conservative initialization
— complexity
— work list based method

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

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

— Rep
— mod
— flow
(e.g

- 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

Algorithm

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

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

— latt
— dat
— val
ts

- 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

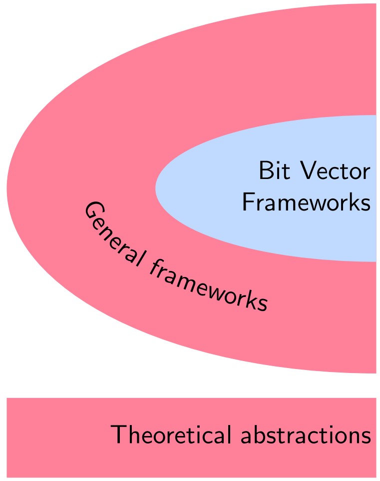
Bit Vector Frameworks

Sequence of Generalizations in the Course Modules

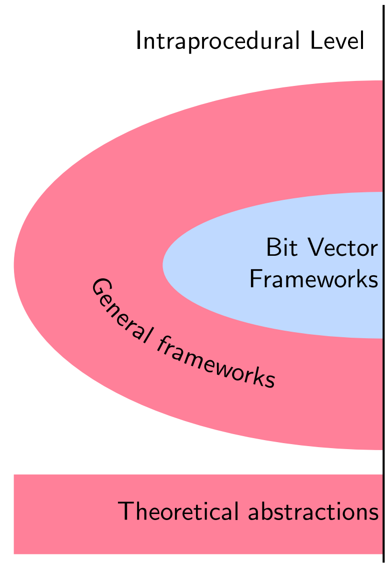
Bit Vector Frameworks

Theoretical abstractions

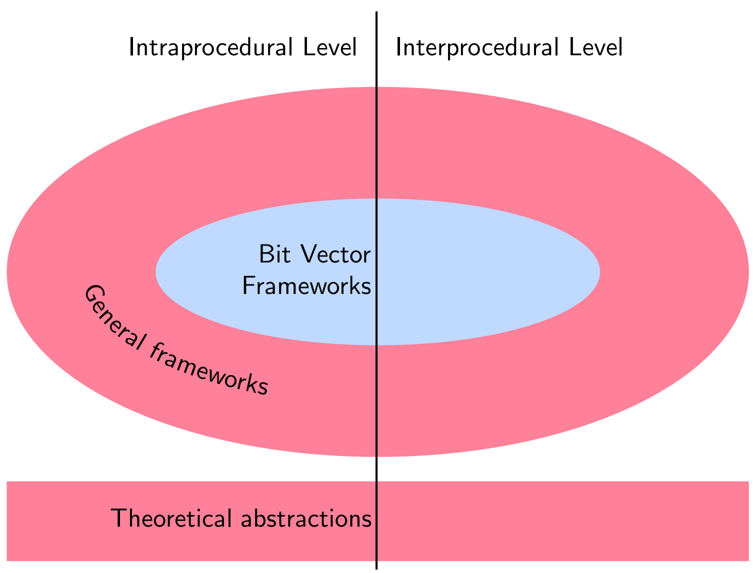
Sequence of Generalizations in the Course Modules



Sequence of Generalizations in the Course Modules



Sequence of Generalizations in the Course Modules



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 essense
  - 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, Vetorization, Dependence analysis
- Other analysis methods
  - Abstract interpretation, Type inference, Constraint resolution

Last But Not the Least

Thank You!