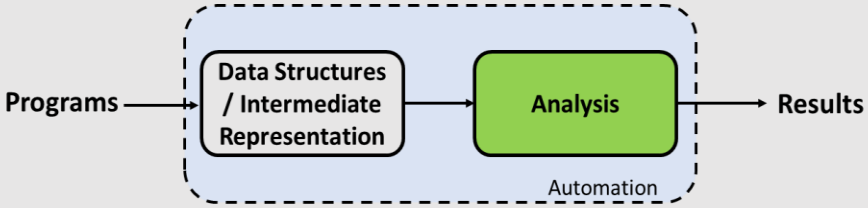
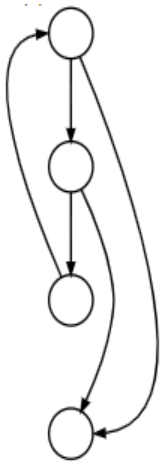
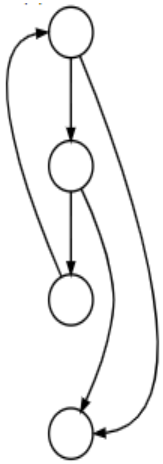




Program Analysis

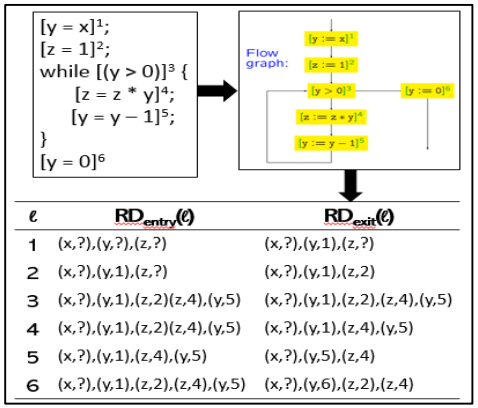
Venkatesh Vinayakarao

venkateshv@cmi.ac.in
Mar – Apr, 2018
Chennai Mathematical Institute



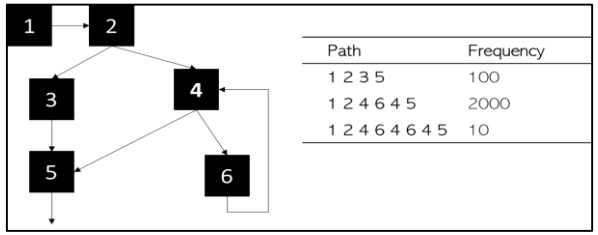
"*Popularity of data-driven software engineering has led to an increasing demand on the infrastructures to support efficient execution of tasks that require deeper source code analysis.*" **– Ganesha and Hridesh, ICSE 2018.**

Quick Review



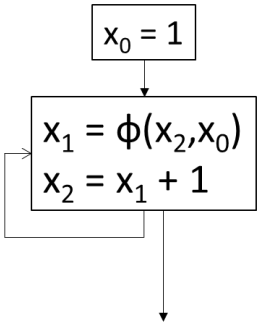
ℓ	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	(x,?),(y,?),(z,?)	(x,?),(y,1),(z,?)
2	(x,?),(y,1),(z,?)	(x,?),(y,1),(z,2)
3	(x,?),(y,1),(z,2)(z,4),(y,5)	(x,?),(y,1),(z,2),(z,4),(y,5)
4	(x,?),(y,1),(z,2)(z,4),(y,5)	(x,?),(y,1),(z,4),(y,5)
5	(x,?),(y,1),(z,4),(y,5)	(x,?),(y,5),(z,4)
6	(x,?),(y,1),(z,2),(z,4),(y,5)	(x,?),(y,6),(z,2),(z,4)

Static Analysis

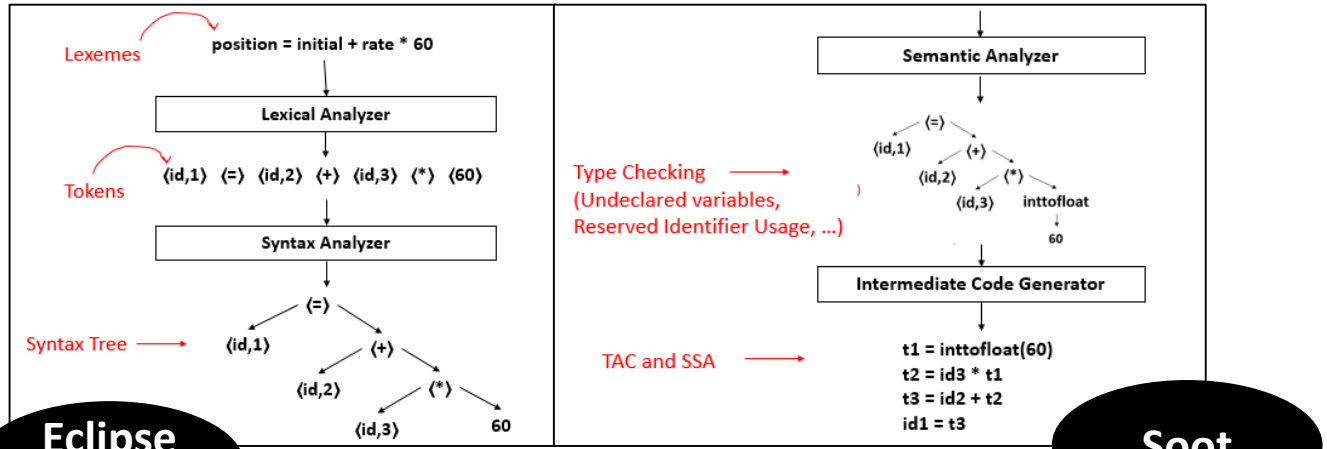


Path	Frequency
1 2 3 5	100
1 2 4 6 4 5	2000
1 2 4 6 4 6 4 5	10

Dynamic Analysis



Graph Representation with SSA Form

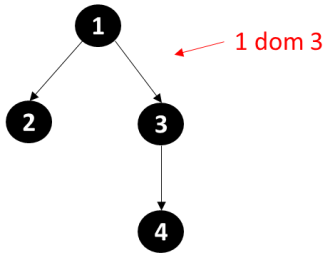


Eclipse JDT

Structure of a Compiler

Soot

jimple



Dominator Tree

Derived Program Representations

Agenda

- Program Representations
  - Basic Blocks
  - Control Flow Graphs
  - Static Single Assignment
  - Three-Address-Code
- Hands-On Session on Traversing a CFG
- More Program Representations
  - Data Dependence
  - Control Dependence
- Dynamic Views of a Program

Assignment-1

Deadline: Submit your code on Moodle by 22nd March 2019 21:00 Hrs. Include a ReadMe.txt file with instructions on how to execute your code.

Data Dependence

S1 **x = y + 10;**

S2 **z = x * y;**

The execution order of S1 and S2 cannot be changed. S2 is data dependent on S1.

Control Dependence Graph

y is control-dependent on x
x determines whether y will be executed

```
S1  if (x > 0) {
S2        y = 10;
S3  }
```

S1 determines whether S2 will be executed. S2 is control dependent on S1.

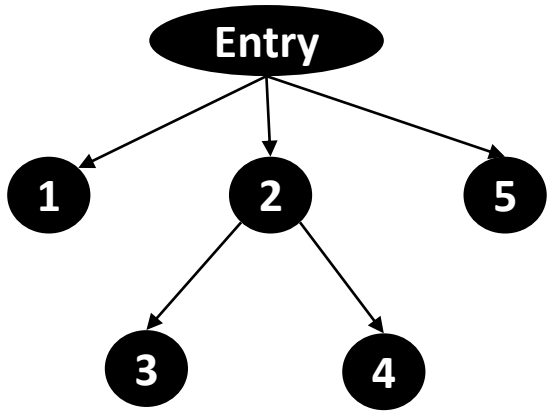
```
S1  while (y > 10) {
S2        y = y + 1;
S3  }
```

S1 determines whether S2 will be executed.
S2 is control dependent on S1.
Also, S1 is control dependent on itself!

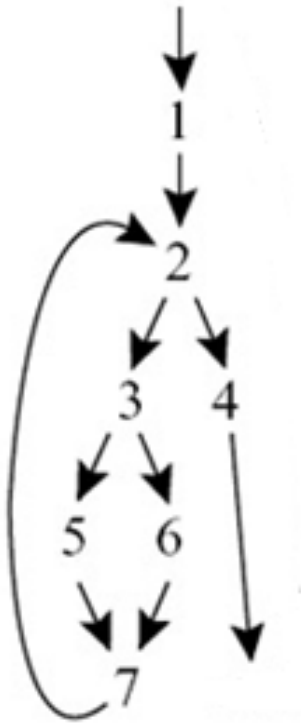
Control Dependence Graph

```
x = 10;
if (x > k)
    y = 10;
else
    y = 1;
print y;
```

```
[x = 10]1;
if [(x > k)]2
    [y = 10]3;
else
    [y = 1]4;
[print y]5;
```

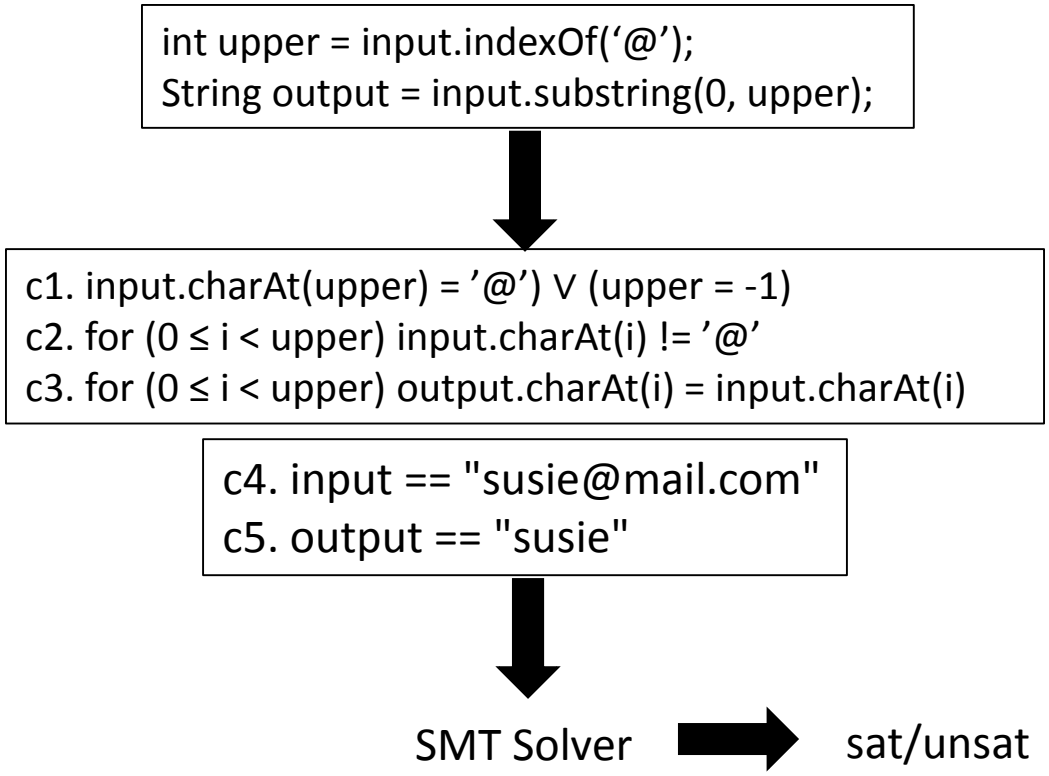


Graded Quiz: Draw the Control Dependence Graph for this CFG



Control Flow Graph

Program as Constraints



Solving the Search for Source Code, Kathryn Stolee,
http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1080&context=computerscidiss

SMT Solvers

- Satisfiability Modulo Theory
  - Input
    - a formula over a background theory
  - Output
    - sat/unsat
      - If sat, it can also compute a satisfying assignment
- We use z3 (SMT) Solver
  - Several theories supported
    - QF_LIA: quantifier free linear integer arithmetic
    - QF_IDL: quantifier free integer difference logic
    - …

Z3 from Java

```
Model check(Context ctx, BoolExpr f, Status sat) throws TestFailedException
{
    Solver s = ctx.mkSolver();
    s.add(f);
    if (s.check() != sat)
        throw new TestFailedException();
    if (sat == Status.SATISFIABLE)
        return s.getModel();
    else
        return null;
}
```

https://github.com/Z3Prover/z3/blob/master/examples/java/JavaExample.java

Readings

- Please read
  - Dragon Book
    - Section 6.2 for Three-Address Code and Static Single-Assignment.
    - Section 8.4.1 and 8.4.3 for Basic Blocks and Flow Graphs.
    - Section 9.6.1 for Dominators and Dominator Tree.
  - Program as Constraints
    - Solving the Search for Source Code, Kathryn Stolee (Optional)

Runtime View of a Program

JProfiler Demo

https://www.ej-technologies.com/blog/2017/03/finding-a-memory-leak-with-jprofiler/

https://www.youtube.com/watch?v=032aTGa-1XM

Animated Bezier Curve Demo - JProfiler 10.0

Session View Profiling Window Help

Start Center | Stop | Save Snapshot | Session Settings — Session

Start Recordings | Stop Recordings | Start Tracking | Run GC | Add Bookmark — Profiling

Export | View Settings | Help | Take Snapshot | Mark Heap | Back | Forward | Go To Start | Show Selection — View specific

- Telemetries
- Live memory
- Heap walker
- CPU views
- Threads
- Monitors & locks
- Databases
- JEE & Probes
- MBeans

JProfiler

No snapshot has been taken.

For a maximum of features:

Press  to take a JProfiler heap snapshot

- The snapshot is displayed in this frame and saved together with profiling information from other views
- For live profiling sessions, special features are available
- Integrations with other views require this snapshot type

Press  to indicate the starting point of a use case

- All objects that are currently on the heap will be marked as old
- When you take the next heap snapshot, new and old objects will be listed separately in the header
- You can select new or old objects only, making it easy to track down memory leaks

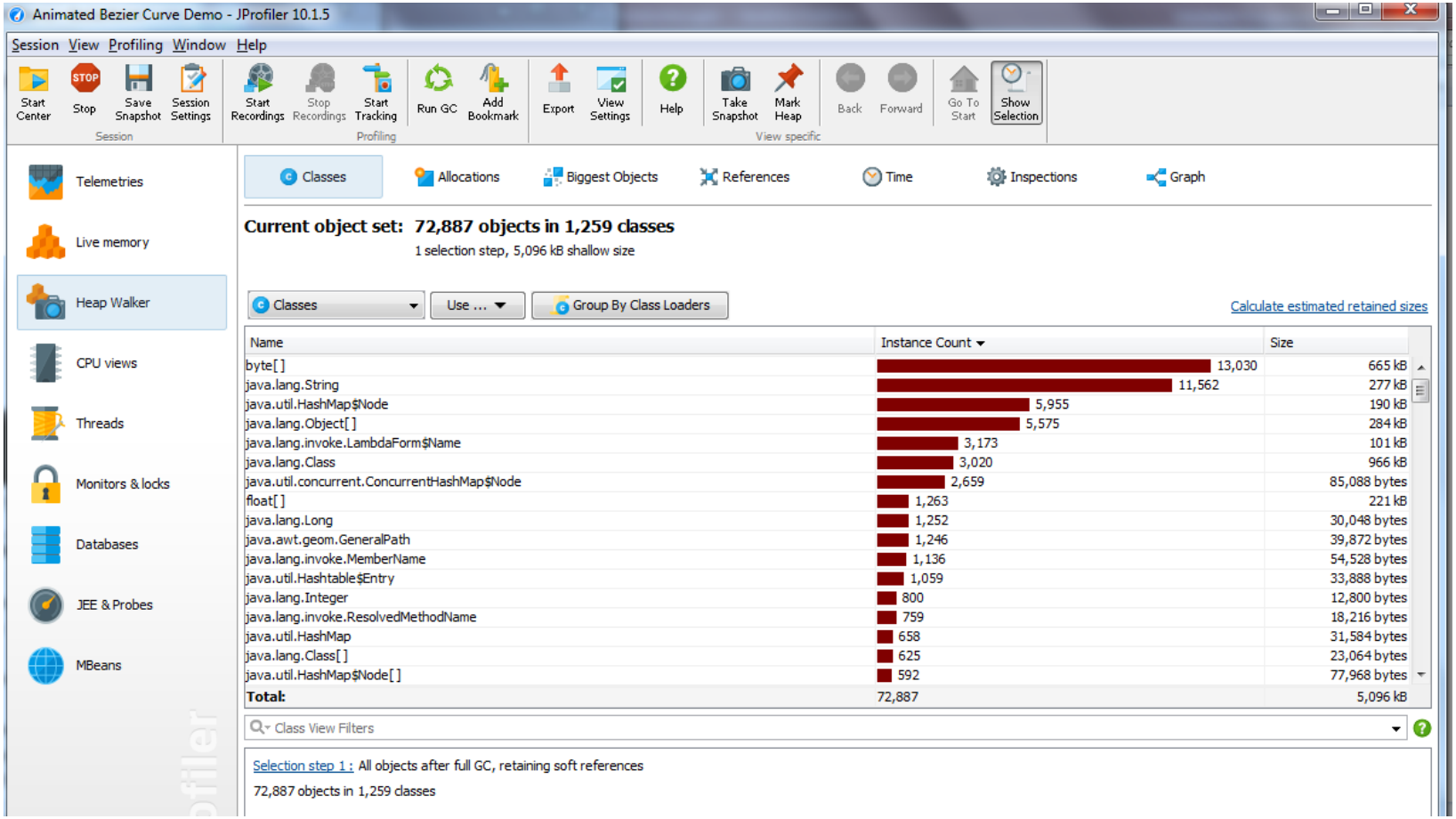
For a minimum of overhead:

Press  to take an HPROF heap snapshot

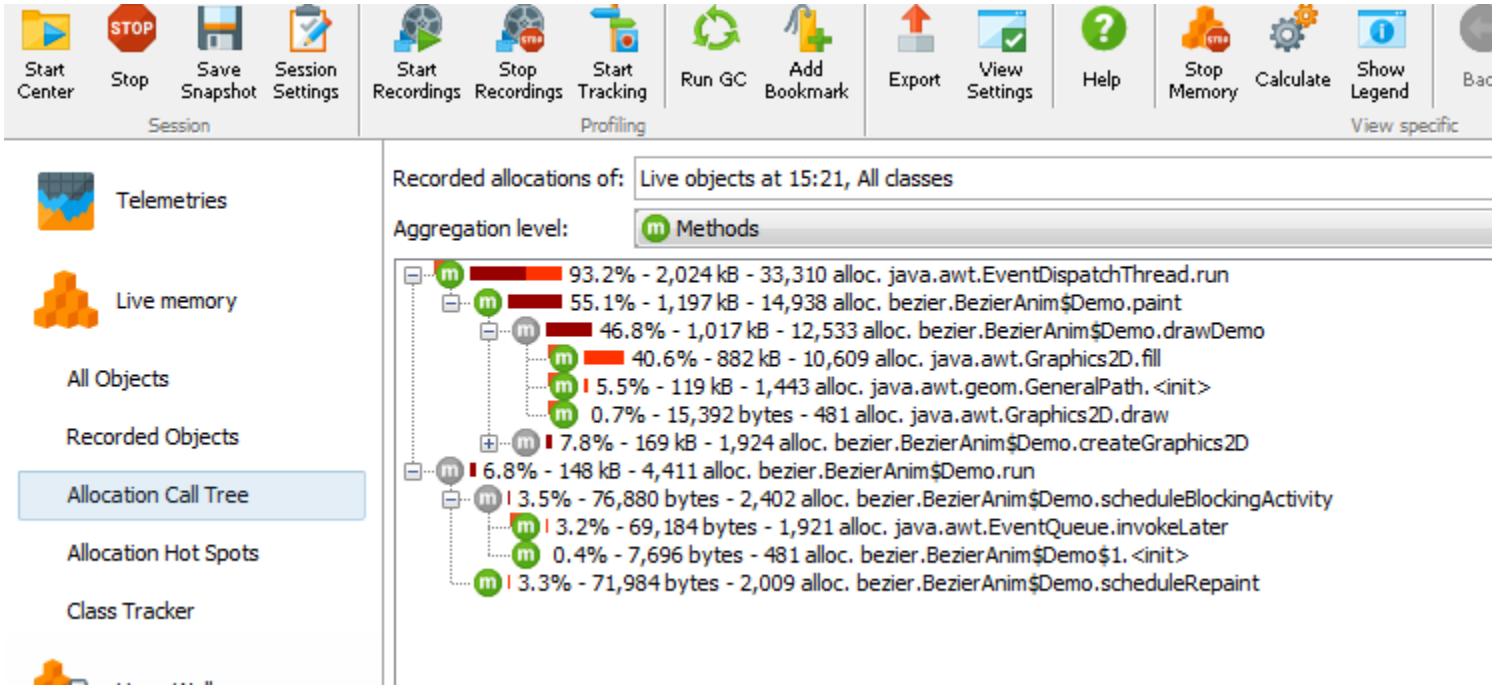
- The snapshot is saved separately and displayed in another frame
- Not all features are available
- Memory and CPU overhead in the profiled VM are lower than for the JProfiler snapshot

0 active recordings | VM #1 | Profiling

Heap Snapshot (using JProfiler)



Call Tree


Start Center
Stop
Save Snapshot
Session Settings
Session
Start Recordings
Stop Recordings
Start Tracking
Run GC
Add Bookmark
Profiling
Export
View Settings
Help
Stop Memory
Calculate
Show Legend
View specific
Telemetries
Live memory
All Objects
Recorded Objects
Allocation Call Tree
Allocation Hot Spots
Class Tracker
Recorded allocations of:
Live objects at 15:21, All classes
Aggregation level:
Methods
93.2% - 2,024 kB - 33,310 alloc. java.awt.EventDispatchThread.run
55.1% - 1,197 kB - 14,938 alloc. bezier.BezierAnim$Demo.paint
46.8% - 1,017 kB - 12,533 alloc. bezier.BezierAnim$Demo.drawDemo
40.6% - 882 kB - 10,609 alloc. java.awt.Graphics2D.fill
5.5% - 119 kB - 1,443 alloc. java.awt.geom.GeneralPath.<init>
0.7% - 15,392 bytes - 481 alloc. java.awt.Graphics2D.draw
7.8% - 169 kB - 1,924 alloc. bezier.BezierAnim$Demo.createGraphics2D
6.8% - 148 kB - 4,411 alloc. bezier.BezierAnim$Demo.run
3.5% - 76,880 bytes - 2,402 alloc. bezier.BezierAnim$Demo.scheduleBlockingActivity
3.2% - 69,184 bytes - 1,921 alloc. java.awt.EventQueue.invokeLater
0.4% - 7,696 bytes - 481 alloc. bezier.BezierAnim$Demo$1.<init>
3.3% - 71,984 bytes - 2,009 alloc. bezier.BezierAnim$Demo.scheduleRepaint

Summary

- Program Representations
  - Lexemes
  - Tokens
  - Abstract Syntax Tree
  - Control Flow Graphs
  - Three-Address Code
  - Static Single Assignment
  - Dominator Tree
  - Data Dependence Graph
  - Control Dependence Graph
  - Program as Constraints
  - Runtime Representations
  - Program Dependence Graph, Hammocks Graphs, State Machines, Petri Nets …