Dynamic Analysis

Rupesh Nasre.

CS6843 Program Analysis
IIT Madras
Jan 2014

Outline

- Applications of dynamic analysis
  - Limitations of static analysis
  - Trade-offs
- Profiling techniques
- Finding invariants
  - Equality
  - Affine
- Dynamic type inferencing

Applications

- Bug finding (testing)
- Data race detection
- Identifying security vulnerabilities
- Improved precision of static analysis
- Input-dependent analysis

Limitations of Static Analysis

- Reduced precision: Over-approximations
- Cannot perform input-dependent analysis

Static versus Dynamic

Static	Dynamic
Sound	Incomplete
Imprecise	Precise
Input-oblivious	Input-dependent

- Choosing between static and dynamic analysis often requires a trade-off between soundness and precision.
- Current trend is to combine the two techniques to get better precision at improved scalability.

Profiling

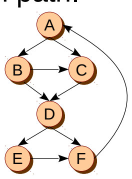
- Profiling is a method of collecting information of interest during program execution.
- The information is often useful to find hot-spots in the program.
- Examples
  - Number of times an instruction is executed
  - Number of page faults
  - Number of cache hits
  - Total memory used
  - ...

Profiling

- Intrusive: inserts instructions in the program (source, IR, assembly) statically, which get executed at runtime
  - File log
  - Memory locations pointed to by a pointer
  - Execution time of a function
- Non-intrusive: the program is unaltered; uses external means to profile
  - Hardware counters
  - Program execution time

Path Profiling

- Consider a program with an entry node and an exit node. There are several execution paths (traces) that the program takes from entry to exit.
- The task is to find the frequency of execution of each path.



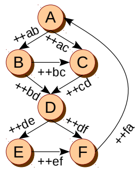
Path	Frequency
ACDF	90
ACDEF	60
ABCDF	0
ABCDEF	100
ABDF	20
ABDEF	0

Path Profiling

- Naïve path profiling is expensive: instrumenting each path may lead to exponential blow up in computation and storage
- This can lead to unacceptable program slowdown

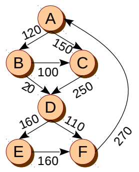
Edge Profiling

- Path profile is approximated as an edge profile
- The frequency of each edge is calculated – which is used to find the path frequency



Edge Profiling

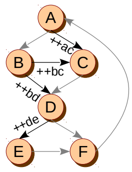
- Path profile is approximated as an edge profile
- The frequency of each edge is calculated – which is used to find the path frequency



Path	Frequency
ACDF	110
ACDEF	150
ABCDF	100
ABCDEF	100
ABDF	20
ABDEF	20

Efficient Edge Profiling

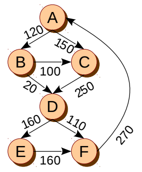
- Observation: We do not need to instrument every edge.
- How to find a minimal, low-cost set of edges to instrument?
- Use a spanning tree: reduced instrumentation along paths, not all edges carry instrumentation



Path	Frequency
C → D	ac + bc
D → F	ac + bc + bd - de
E → F	de
A → B	bc + bd
F → A	ac + bc + bd

Edge Profiling

- Edge profile may not always be a good indicator of a path profile
- Efficient edge profiling requires a unique variable along each instrumented edge (spanning tree edge)

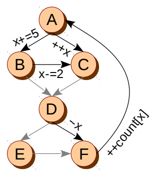


Path	Frequency	Actual Freq.	Actual Freq. 2
ACDF	110	90	110
ACDEF	150	60	40
ABCDF	100	0	0
ABCDEF	100	100	100
ABDF	20	20	0
ABDEF	20	0	20

But path profiling is expensive

Efficient Path Profiling

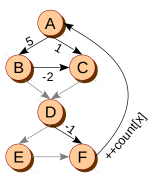
- Unique (and consecutive) path numbering, which enables indexing
- Most hardware support fast increment and indexing

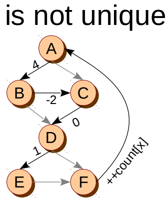


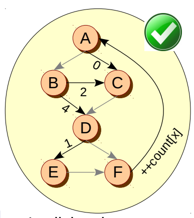
Path	x
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Efficient Path Profiling

- Path numbering is not unique







Path	x
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

In all the above cases, the path numbering is the same, number of instrumented edges (5) is the same

So, which instrumentation should we choose?

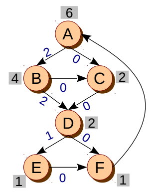
Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

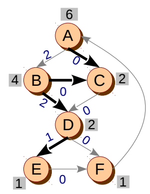
```
NumPaths(node) = 0
NumPaths(leaf) = 1
In reverse topological order
  For each edge v → w {
    Val(v → w) = NumPaths(v)
    NumPaths(v) += NumPaths(w)
  }
```



Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

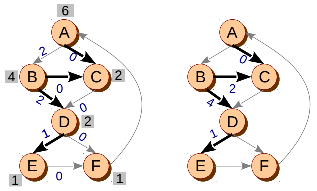
- Find a spanning tree.
- Find **chord** (non-ST) edges.
- For each chord, find fundamental cycle.



Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

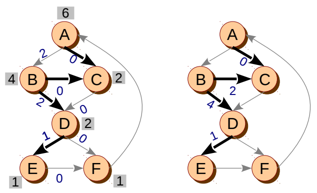
Chord AC: cycle ACDF : 0
Chord BC: cycle ABCDF : 2
Chord BD: cycle ABDF : 4
Chord DE: cycle DEF : 1



Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Chord AC: cycle ACDF : 0
Chord BC: cycle ABCDF : 2
Chord BD: cycle ABDF : 4
Chord DE: cycle DEF : 1



Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Prelude: Allocate and initialize the array of counters

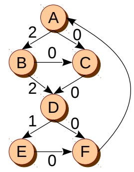
Postlude: Write the array to permanent storage

Main:
- Initialize path register r in the entry vertex
- Increment path memory counter in the exit vertex
- Optimizations

Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Path Regeneration
Path id → Path mapping?



Path	id
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5