The Essence of Dynamic Analysis

Thomas Ball
Microsoft Research
(modified by Zhang)

A "Present" Challenge for Dynamic Analysis

```c
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
}
```

Pretty Printed Code

```
#include <stdio.h>
main(t,_,a)
     char *a;
{
  if ((!0) < t) {
    if (t < 3)
      main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a));
    if (t < _ )
      main(t+1,_,a);
    if (main(-94,-27+t,a)) {
      if (t==2 ) {
        if ( _ < 13 ) {
          return main(2,_+1,"%s %d %d\n");
        } else {
          return 9;
        }
      } else
        return 16;
    } else
      return 0;
...
```

A Folk Theorem

- any program can be transformed into a semantically equivalent program consisting of a single recursive function containing only conditional statements

The Most Basic Dynamic Analysis: Run the Program!

```
On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

...

On the twelfth day of Christmas my true love gave to me
twelve drummers drumming, eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.
```

The Output Pattern

- On the *<ordinal>* day of Christmas my true love gave to me *<list of gift phrases, from the ordinal day down to the second day>* and a partridge in a pear tree.
- The first verse:
  - On the first day of Christmas my true love gave to me a partridge in a pear tree.

Modelling of the "12 Days" with Frequencies

- 12 days of Christmas
- 26 unique strings
- 66 occurrences of non-partridge-in-a-pear-tree gifts
- 114 strings printed
- 2358 characters printed

File Profile View Go Bookmarks Options

Open Filter Close Home Back Forward Add Bookmark Bookmarks

Browser View | Source View

Path Id	Procedure Name	Frequency	Length	Number of Instructions
9	main	1	67	67
	main	1	27	27
2	main	1	67	67
3	main	10	74	740
	main	11	35	385
3	**main**	**55**	**42**	**2310**
	main	114	27	3078
	main	114	28	3192
	main	2358	43	101394
	main	2358	56	132048
	main	24931	39	972309
	main	39652	39	1546428

Paths

File: transformed.c

```
#include <stdio.h>
main(t,_,a)
    char *a;
{
  if ((!0) < t) {
    if (t < 3)
      main(-79,-13,a+main(-87,1-_

    if (t < _ )
      main(t+1,_,a);

    if (main(-94,-27+t,a)) {
      if (t==2 ) {
        if ( _ < 13 ) {
          return main(2,_+1,"%s %d %
        } else {
          return 9;
        }
      } else
        return 16;
    } else
      return 0;

  } else if (t < 0) {
    if (t < -72) {
```

12 days of Christmas

26 unique strings

66 occurrences of non-partridge-in-a-pear-tree gifts

114 strings printed

2358 characters printed

Other Examples of Dynamic Analyses

- Program Hot Spots
- Memory Reference Errors
  - uninitialized memory, segment fault and memory leak errors
- Coordination Problems
  - racing data accesses in concurrent programs
- Security of Web Applications
  - tainted values

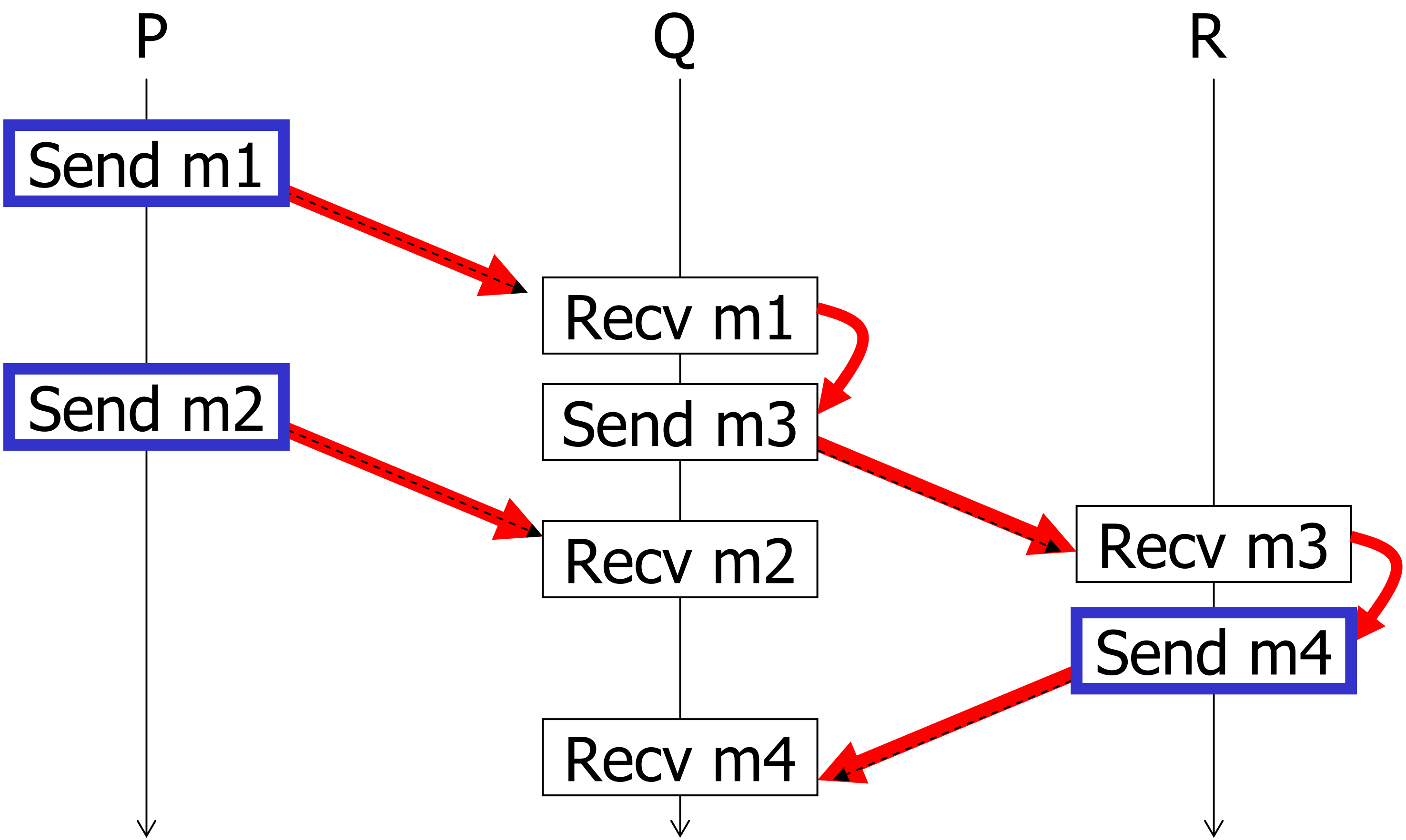
Program Hot Spots

- How many times does each program entity execute?
  - Procedures, methods, statements, branches, paths
- 80-20 rule
  - 20% of program responsible for 80% of execution time
- Applications
  - Performance tuning
  - Profile-driven compilation
  - Reverse engineering

Memory Reference Errors

- Purify, a popular link-time instrumentation tool, detects
  - reads of uninitialized memory
  - accesses to deallocated memory
  - accesses out of bounds
- Memory instrumentation via memory map
  - 2 bits per byte of memory
    - allocated, uninitialized, initialized
    - "red zone"
- Purify substitutes its own malloc; each load/store instrumented to test/set bits

Race Condition Detection



[Netzer, Miller

Secure Web Applications

- Perl
  - popular interpreted scripting language used for many tasks, including CGI programming
- “tainted” Perl
  - each scalar value received from the environment is “tainted”
  - “tainted” values propagate through expressions, assignment, etc.
  - “tainted” values cannot be used in critical operations that can write to system resources

Outline

- What is dynamic analysis?
  - Example: path profiling
- How is it accomplished?
  - Precision vs. Efficiency
- Relationships to static analysis
- Trends

What is Dynamic Analysis?

Dynamic analysis is the investigation of the properties of a running software system over one or more executions

What is Dynamic Analysis?

- What is the meaning of “run”?
  - abstract interpretation and static analyses “run” a program over an abstract domain
  - OUT=F(IN,s)
- Dynamic analysis
  - abstraction used in parallel with, not in place of, concrete values
  - OUT=F(IN, s_i, v)

Some Characteristics of Dynamic Analysis

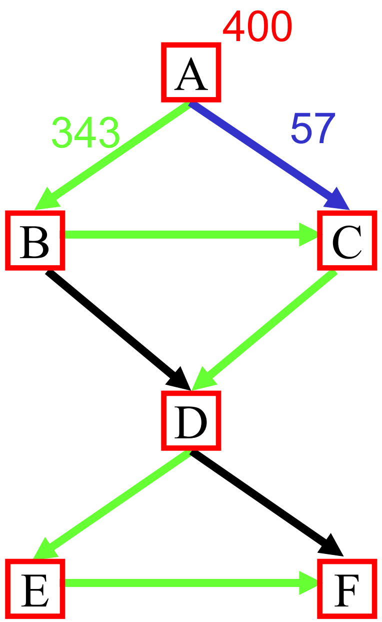
- Dynamic analysis can collect exactly the information needed to solve a problem
  - Procedure specialization: parameter values
  - Dynamic program slicing: flow dependences
  - Race conditions: message sends
- Scales very well
- Can be language independent!
  - Record information at interfaces

Fundamental Results in Dynamic Analysis

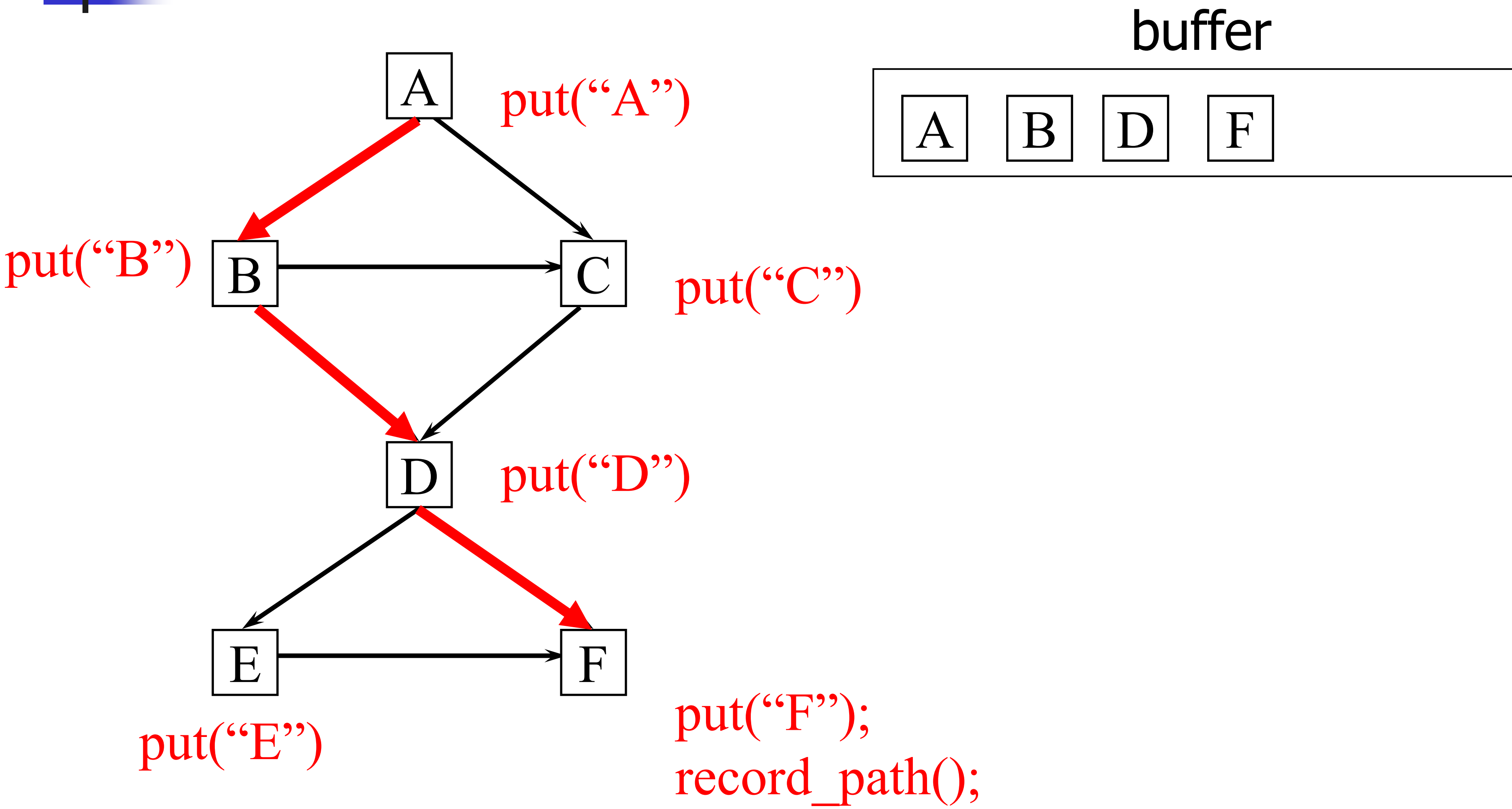
- Dynamic analysis is, at its heart, an experimental effort
  - Have insight
  - Build tool
  - Evaluate efficiency and effectiveness
  - Rethink

Example: Path Profiling

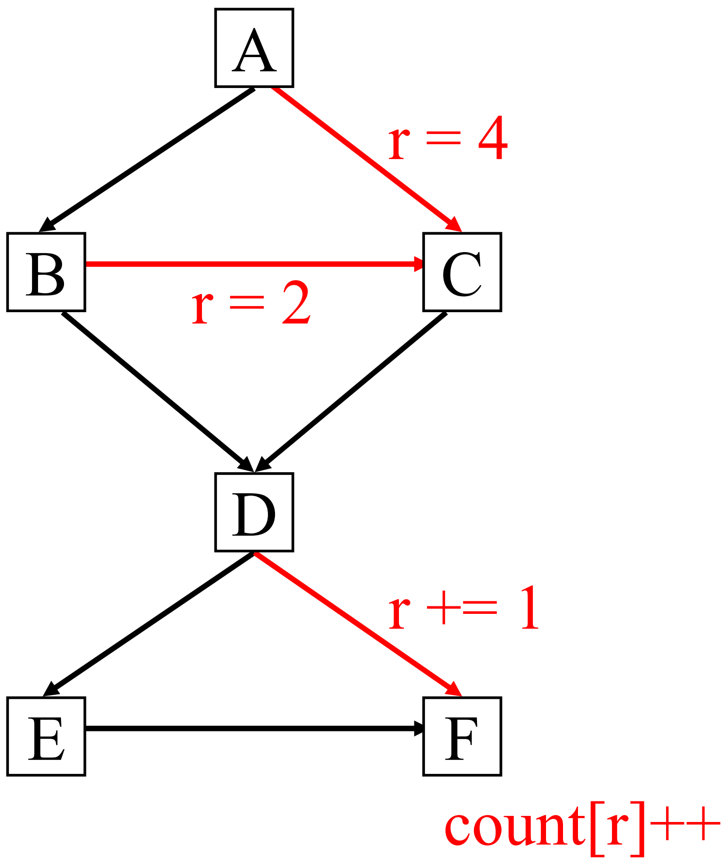
- How often does a control-flow path execute?
- Levels of profiling:
  - blocks
  - edges
  - paths



Naive Path Profiling

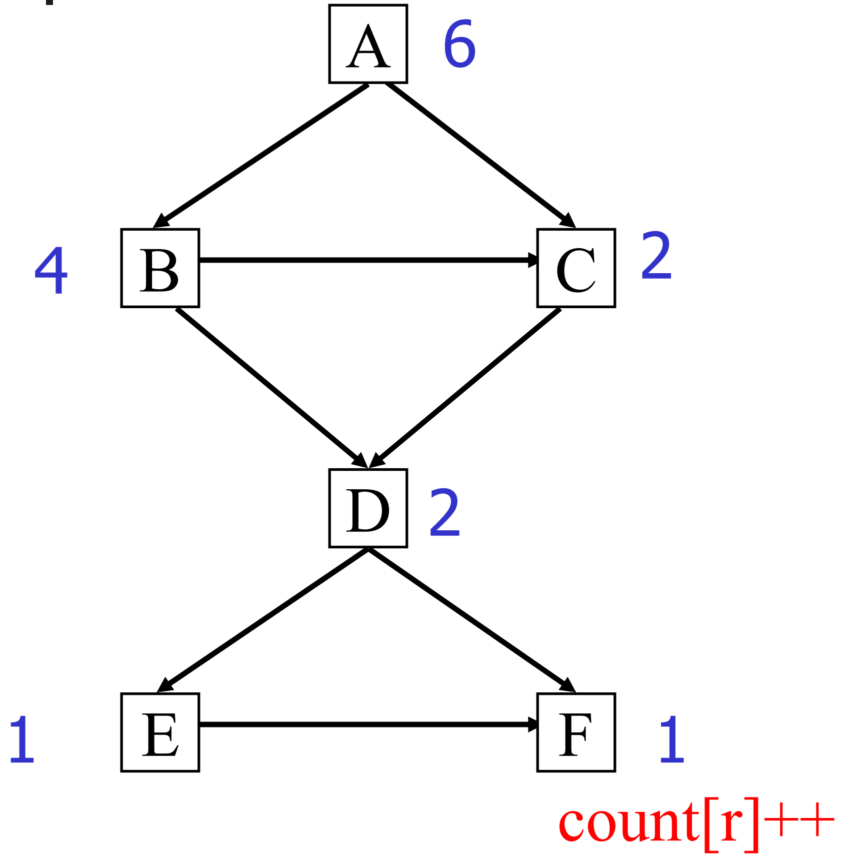


Efficient Path Profiling

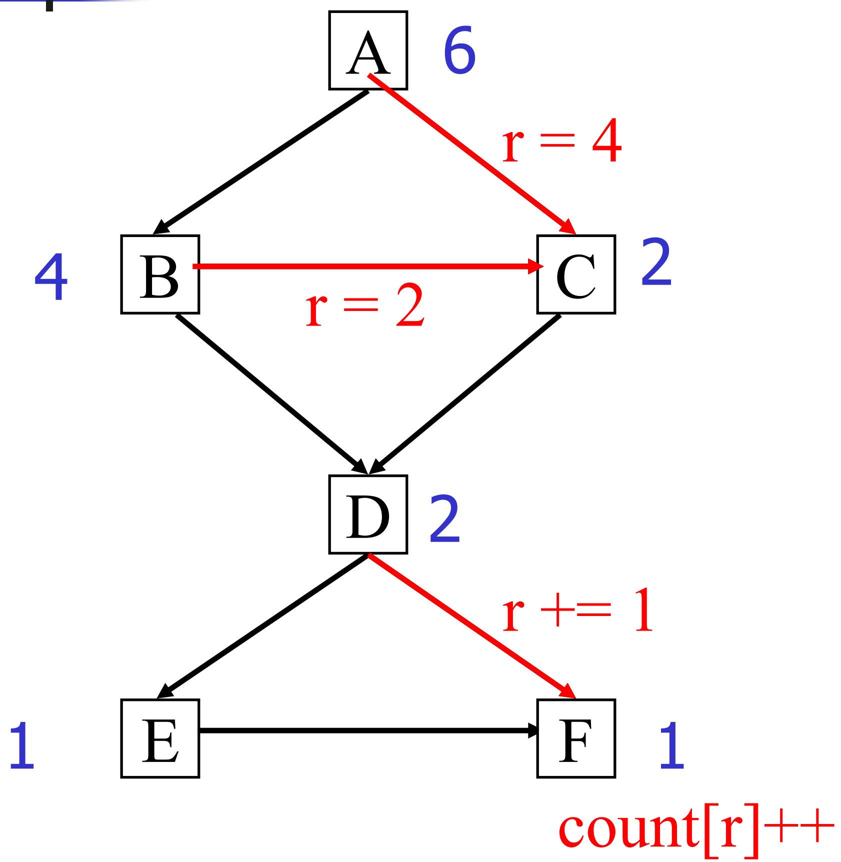


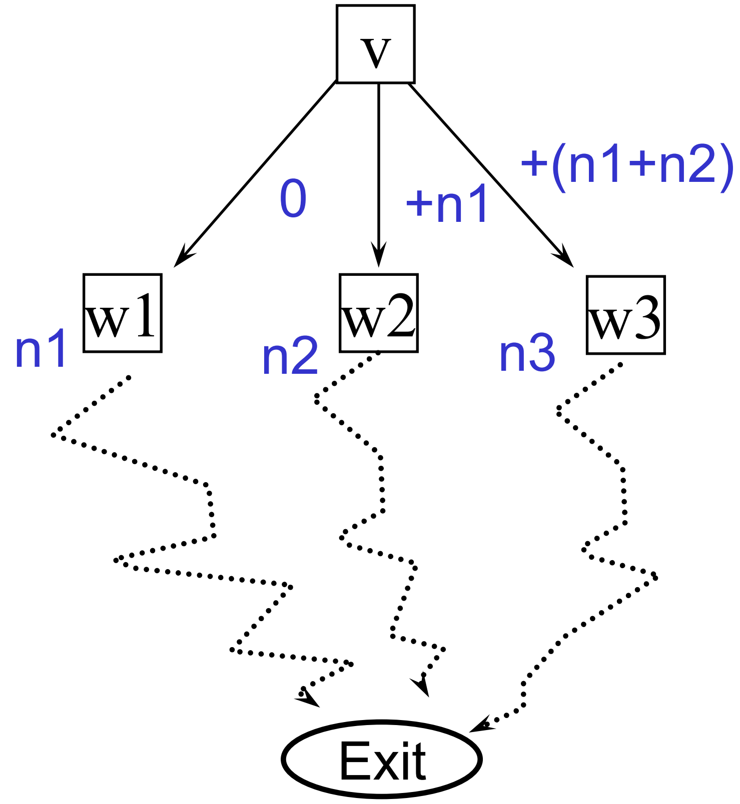
Path	Encoding
ABDEF	0
ABDF	1
ABCDEF	2
ABCDF	3
ACDEF	4
ACDF	5

Efficient Path Profiling



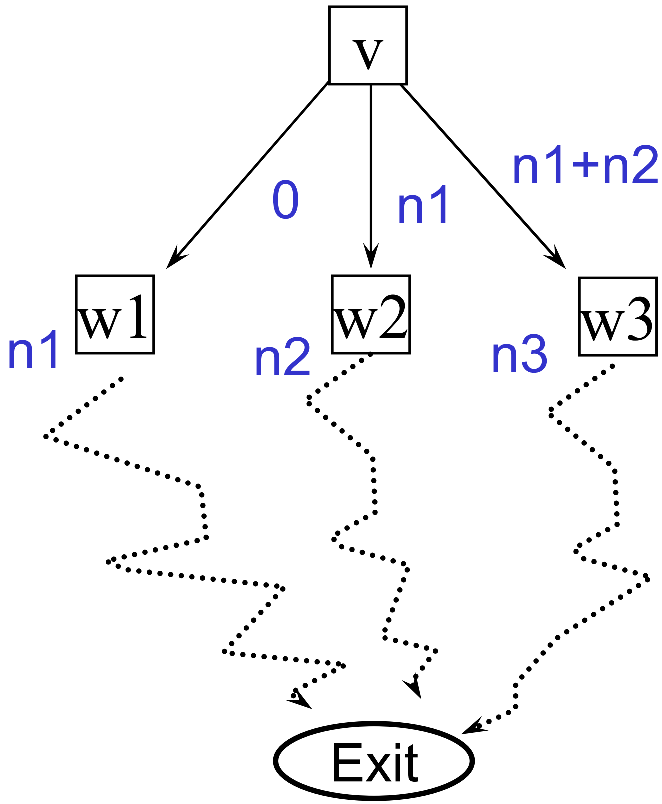
Efficient Path Profiling

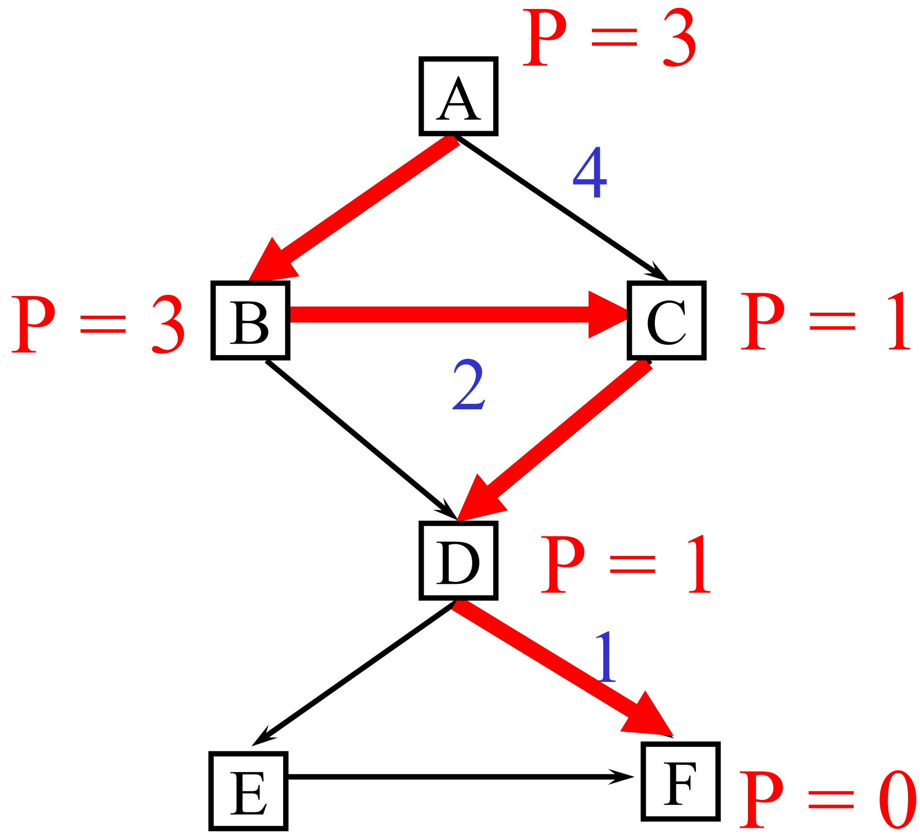




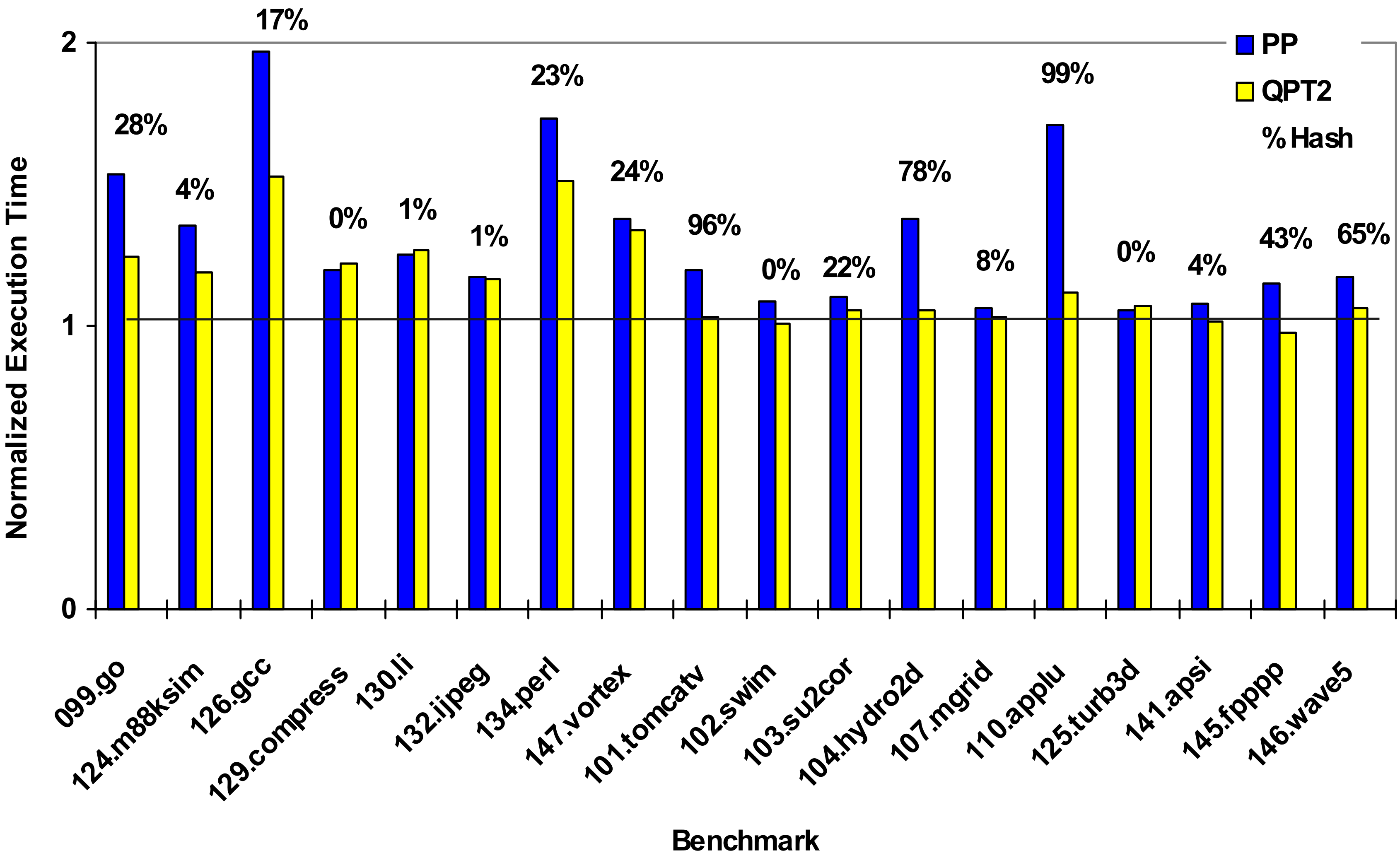
Path Regeneration

- Given path sum P, which path produced it?

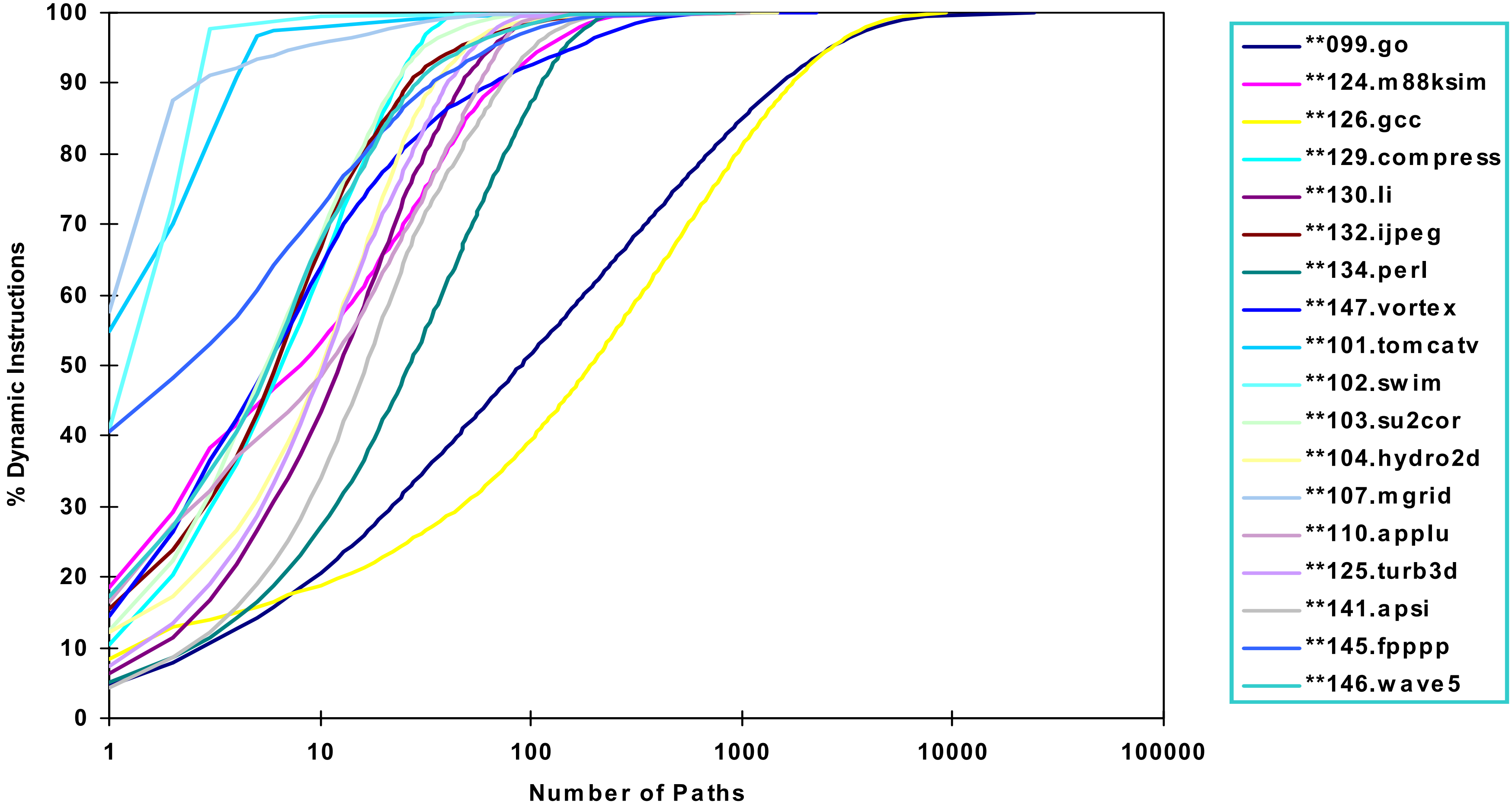




PP Efficiency



Effectiveness



Aggregation and Compression

- Dynamic analysis is a problem of data aggregation and compression, as well as abstraction
  - frequencies vs. the full trace
    - Efficient path profiling relies on cutting full trace into shorter paths
      - Makes analysis efficient
      - Loses loop and procedural contexts
  - If full trace, how to compress
    - Zlib, sequittur, bdd, value predictor, WET…
    - Execution reduction, check pointing
  - Abstraction
    - Purify uses two bits per byte of memory

Outline

- What is dynamic analysis?
- How is it accomplished?
  - Precision vs. Efficiency
- Relationships to static analysis, model checking, and testing
- Trends

How is Dynamic Analysis Accomplished ?

- Observation of behavior
  - hardware monitoring
  - PC sampling
  - breakpoints
- Instrumentation
  - code added to original program
  - ideally does not affect semantics of program
  - does affect the running time of a program
- Interpreters
  - interpreter instrumentation

Creating Instrumentation Tools

- Source-level
  - Pattern-matching over parse tree or AST and rewriting
  - A* [Ladd, Ramming], Astlog [Crew], ...
  - Full access to source information and precise mapping
- Binary
  - ATOM [Srivastava] , EEL [Larus], Diablo, Bluto...
  - Analyze programs from multiple languages
  - Limited access to source information
- Run-time
  - Valgrind, PIN

Instrumentation Issues

- How much to generate?
  - Everything
  - Just the necessary facts
  - Less than necessary
- On-line vs. off-line analysis
- What/When to instrument?
  - Source code, IR, assembly, machine code
  - Preprocessor, compile-time, link-time, executable, run-time
- Automation

Outline

- What is dynamic analysis?
- How is it accomplished?
  - Precision vs. Efficiency
- Relationships to static analysis
- Trends

Static and Dynamic Analysis, Explained

Program + Input = Behavior

Static Analysis

Program + ~~Input~~ = Behavior

Program as a guide to behavior

- input insensitive

Dynamic Analysis

Program + Input = Behavior

Input + behavior as a guide to the program

- Input sensitive

Dynamic and Static Analysis

- Completeness
  - static complete
  - dynamic incomplete
- Precision
  - dynamic analysis can examine exactly the concrete values needed to help answer a question
    - All state along one/a few paths.
  - static analysis confounded by abstraction and infeasible paths
    - A small subset of states for all possible paths

Diving Deeper…

- Abstraction
- Infeasible paths
- Interplay between static and dynamic analyses

Abstraction

- Static analysis
  - abstraction is required for termination
    - Bound number of states (stores)
    - Bound size of each state (store)
- Dynamic analysis
  - termination is a property of the running system, not a major concern of analysis
  - abstraction helps reduce run-time overhead
    - Purify: two bits per byte to record state of memory
    - Path profiling: short paths rather than long traces
- Precision a concern in both

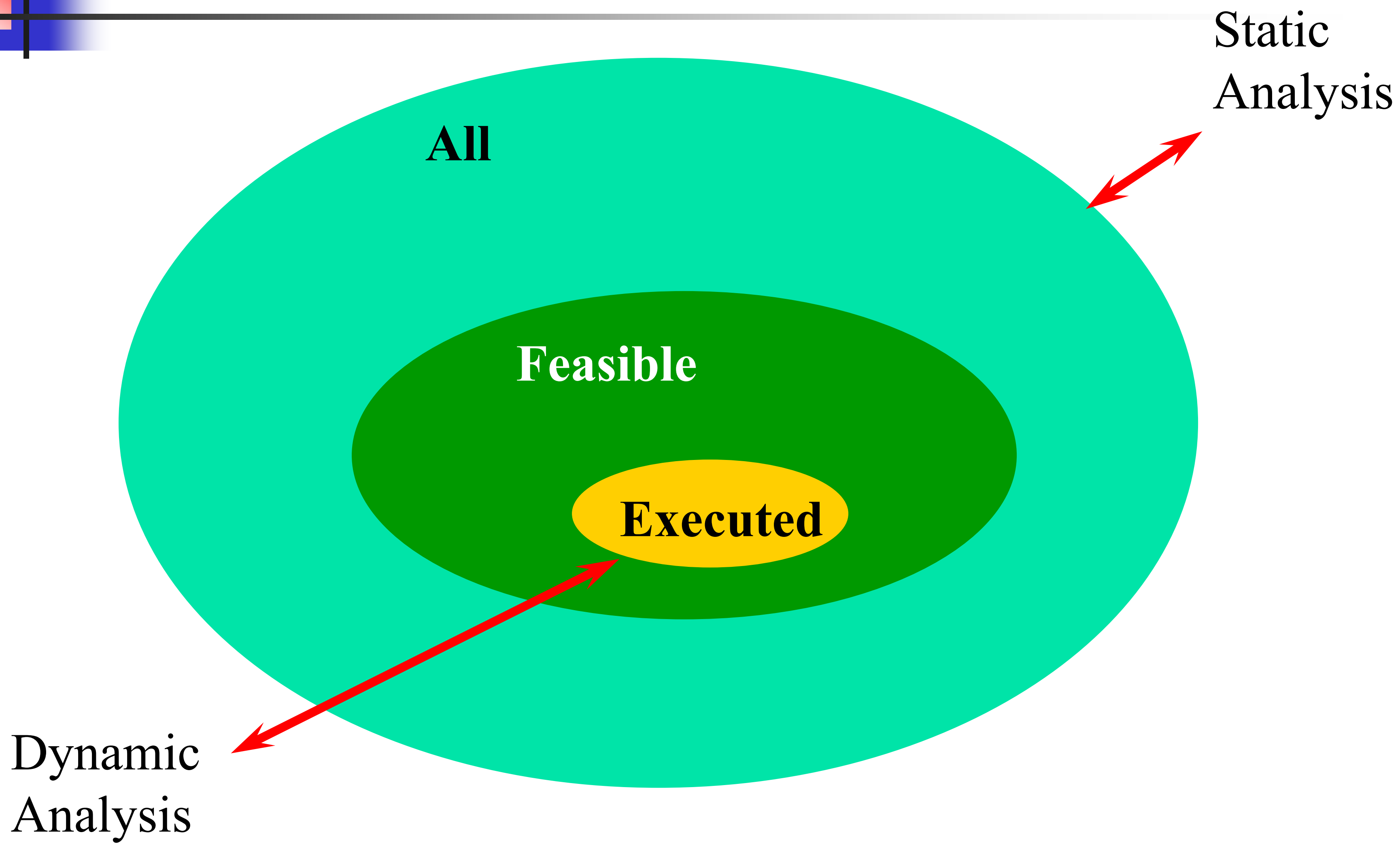
Feasible and Infeasible Paths

- Dynamic analysis leaves feasible paths unexplored
  - may conclude a property holds when it really doesn't (precise for test set but unsafe)
- Static analysis explores infeasible paths
  - may conclude a property doesn't hold when it really does (safe but imprecise)
- What can one do to increase confidence in either analysis?

```
Node* Delete(Node* z) {
  Node *y, *x;

  if ((z->left == nilNode) ||     [36]
      (z->right == nilNode))
    y = z;
  else
    y = treeSuccessor(z->right);

  if (y->left != nilNode)         [12]
    x = y->left;
  else
    x = y->right;

  x->parent = y->parent;

  if (y->parent == nilNode)        [6]
    root = x;
  else if (y == y->parent->left)
    y->parent->left = x;
  else
    y->parent->right = x;

  if (y != z)                      [2]
    z->key = y->key;

  return(y);
```

- 36 total paths
- 8 feasible paths
- False flow dependences

Control Flow Paths



Two Sides of Imprecisoin

- Imprecision in Dynamic Analysis
  - (Feasible-Executed)/Feasible
  - increase precision as Executed approaches Feasible
    - systematic generation of tests
- Imprecision in Static Analysis
  - (All-Feasible)/All = Infeasible/(Infeasible+Feasible)
  - increase precision as Infeasible approaches 0
    - methods to eliminate infeasible paths

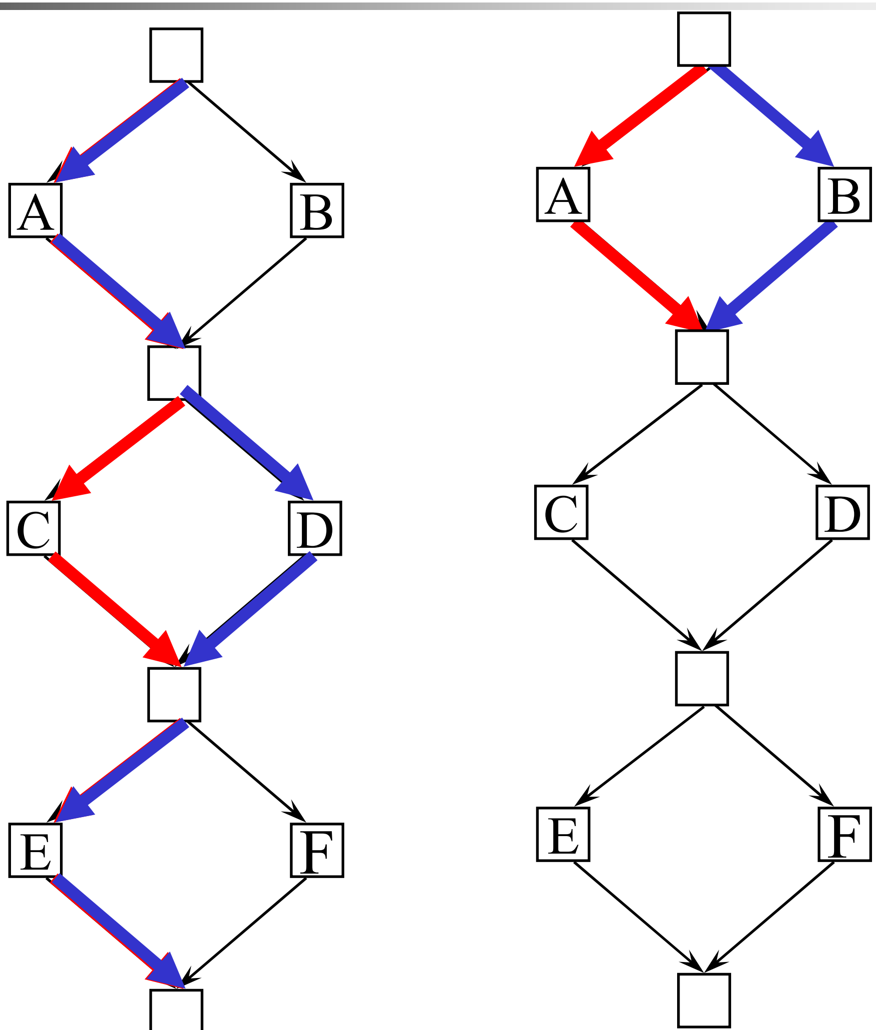
```
de* Delete(Node* z) {
Node *y, *x;

if ((z->left == nilNode) ||   [36]
     (z->right == nilNode))
  y = z;
else
  y = treeSuccessor(z->right);

if (y->left != nilNode)       [12]
  x = y->left;
else
  x = y->right;

x->parent = y->parent;

if (y->parent == nilNode)      [6]
  root = x;
else if (y == y->parent->left)
  y->parent->left = x;
else
  y->parent->right = x;

if (y != z)                    [2]
  z->key = y->key;

return(y);
```

```
Node* Delete(Node* z) {

  if (z->left == nilNode)          [9
    return reparent(z,z->right);
  else if (z->right == nilNode)    [6
    return reparent(z,z->left);
  else {                           [3
    Node *y = treeSuccessor(z->right
    z->key = y->key;
    return reparent(y,y->right);
  }

Node* reparent(Node *n, Node *c) {
  c->parent = n->parent;
  if (n->parent == nilNode)        [3
    root = c;
  else if (n == n->parent->left)   [2
    n->parent->left = c;
  else                             [1
    n->parent->right = c;
  return n;
}
```

State Space

- Dynamic and static analysis represent two extremes of state space exploration of programs
- Dynamic analysis is a depth-first exploration of program behavior
- Static analysis is “breadth-first”, sort of…
  - combines information from multiple paths
  - the longer the paths analyzed, the greater the chance that results will be imprecise
    - infeasible paths
    - abstraction

Program Paths

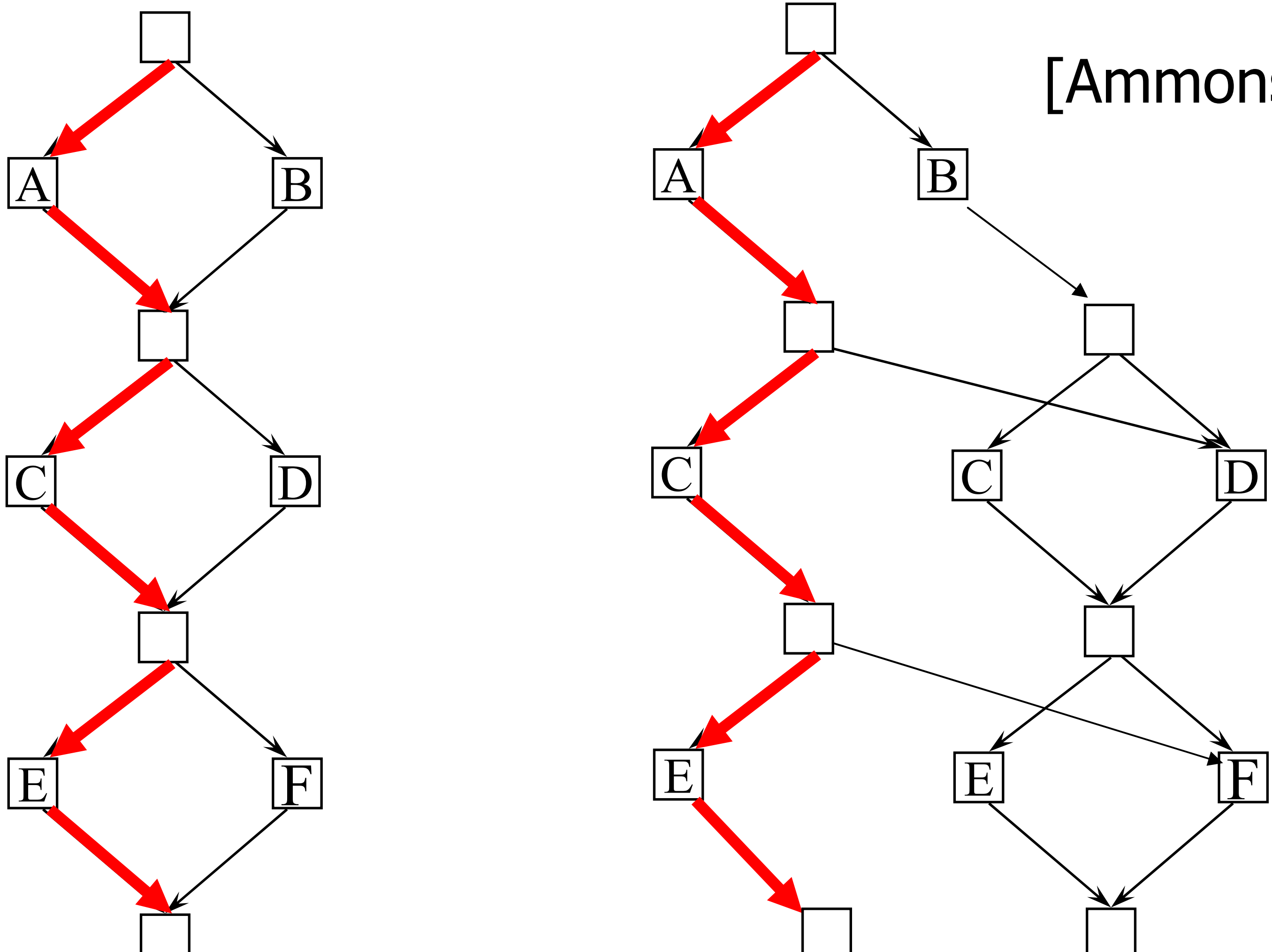


Interplay of Dynamic and Static Analysis

- Data Flow Analysis
  - path-sensitive DFA
  - “widening” DFA
- Program Slicing

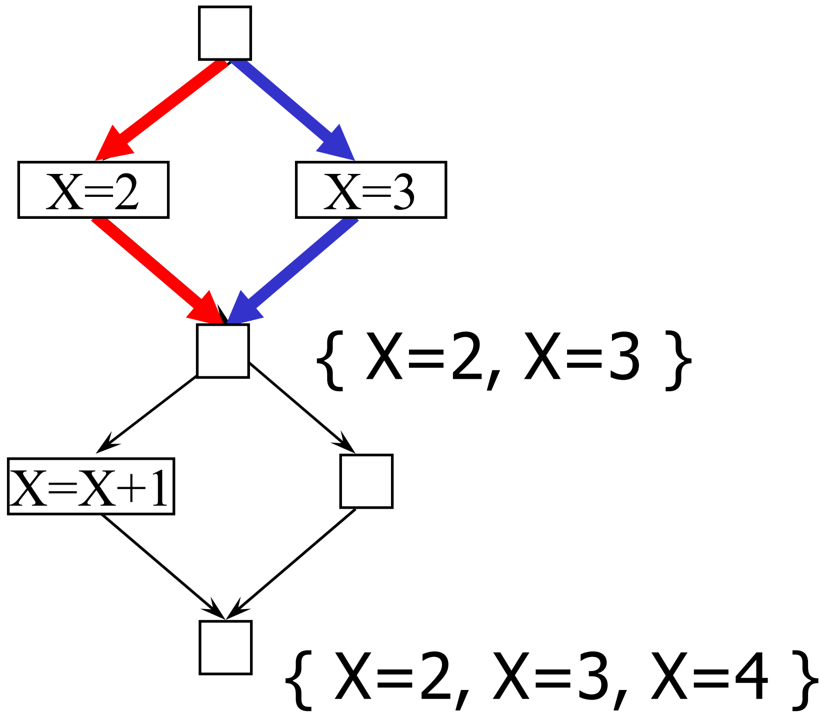
Restructuring for Path-sensitive Data Flow

[Ammons, Larus



"Widening" Data Flow Analysis

- Keep info at merge rather than lose
  - collecting semantics
- Can't collect everything
  - What to keep, what to drop?



Program Slicing

- Static Analysis
  - Control flow analysis
  - reaching definitions
  - pointer alias and shape analysis
- Dynamic Analysis
  - exact computation of flow dependences in trace

Dynamic/Static Analysis for Slicing

- Levels of precision
  - Compute flow dependences between statement instances
  - Compute paths/edges/nodes covered and perform static analysis over these entities

[Agrawal, Horgan]

Outline

- What is dynamic analysis?
- How is it accomplished?
  - Precision vs. Efficiency
- Relationships to static analysis, model checking, and testing
- Trends

Size and Complexity

- Plagues both static and dynamic analyses, though less for the latter
  - State space and path explosion for static analysis
  - Depth-first scales

Binding times

- Binding times of program and system components are becoming more and more dynamic
  - Virtual functions,Factories, Objects, DLLs, Dynamic class loaders, …
  - Boon to extensibility, reconfigurability, maintenance
  - A thorn for static analysis

Multi-lingual Systems

- How many languages does it take to deploy a web application?
  - Client side
    - HTML, Java
  - Server side
    - A general purpose language: Perl, C, C++, Java, …
    - Server side scripting: Javascript, ASP, …
    - Database languages: SQL
- Tcl and integrating applications
- How to analyze a system in the face of multiple languages?
  - Will analysis at the interfaces suffice?

A Golden Age for Dynamic Program Analysis

Open Problems

- The problem of perturbation
- Dynamic differencing
- Dynamic analysis and test generation
- Frameworks for dynamic analysis
- Interactions of dynamic analysis, languages and optimizations
- Machine learning models of program behavior
- Hybrid dynamic/static analyses
- Analyzing non-terminating programs