Dynamic Program Analysis

Modified from T. Ball's slides
(Microsoft Research)

A "Present" Challenge for Dynamic Analysis

```c
#include <stdio.h>
main(t,...,a)
    char *a;
{ return !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?c13?
    main(2,...,"%s %d %d
":9:16:t<0?t<-72?main(_,t,
    "@n'+'/*{w+w/w/cdnr/+,(r/*de)+,/*+,w+/w/q#n+,#(l+,n(n+,+/q+n+,/#
    /q+n+,+/k#:*e,'r:'d*3,)w+wK'k':+e:}'dq#"1 
    q#'d'+k#:q#'z)eKK#w'z)eKK[nl]'#/gq#n'}{(w'}){(nl)'+/q#dzw' i:# 
    }{nl}'/n(n'; r(##r nc[nl]'#/l+,K [rw' iK[]{[nl]'/w#q#n'}).wK ne' 
    iwK[KK[nl]'/w t1##w'; i: :{nl}'/*q#'ld;z'}{nlw'b/*de'}c 
    ;:nl'{-[rw']+/j##*#nc,'#nw'+kd'+'#zdq#w! nz'/ '})++}{zl'{'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,*,
    '@ek;dc hicK'q)-[w]*nz+1,{},m.wloca=;m .vpkfs,fntcdCeghiy"},a+1);
}
Pretty Printed Code

```c
#include <stdio.h>
main(t,_,a)
  char *a;
  {
    if (!(t)< 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
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

The diagram illustrates the sequence of events in a race condition scenario. Each node represents a process or event, and the arrows indicate the flow of execution. The processes are labeled as `-P`, `-Q`, and `-R`, with events such as `Send m1`, `Send m2`, `Recv m1`, `Recv m2`, `Send m3`, `Recv m3`, and `Send m4`. The diagram highlights the potential race condition where events may occur in an unpredictable order, leading to unexpected outcomes.

### Secure Web Applications

- **Perl**
  - A 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.

- [Netzer, Miller]
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, si, 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

```
put("A")
B

put("B")
C

put("C")
D

put("D")
E

put("E")
F

```

Efficient Path Profiling

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

```cpp
r = 4
count[r]++
```
Efficient Path Profiling

```
count[r]++
r = 4
Exit
r += 1
```

```
count[r]++
r = 4
r = 2
r += 1
```

```
r = 2
+(n1+n2)
+ n1
```

```
0
+n1
```

```
Exit
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```

```
A 6
B 4
C 2
D 2
E 1
F 1
```
Path Regeneration

Given path sum \( P \), which path produced it?

- \( P = 3 \)
- \( P = 1 \)
- \( P = 0 \)

PP Efficiency

Benchmark: various programs with different execution times and normalized execution times. The graph compares PP and OPT2 efficiencies, showing a percentage decrease in execution time for each benchmark.
Effectiveness

<table>
<thead>
<tr>
<th>% Dynamic Instructions</th>
<th>Number of Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>1000</td>
</tr>
<tr>
<td>40</td>
<td>10000</td>
</tr>
<tr>
<td>50</td>
<td>100000</td>
</tr>
</tbody>
</table>

Comments

- Limitations
  - Can not go beyond loop edges and call edges
  - Path explosion entails hashing

- Attractions
  - Number interprocedural paths
  - Solve path explosion
    - Path poisoning
    - New encoding
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 and virtual machines

Creating Instrumentation Tools

- Source-level
  - Pattern-matching over parse tree or AST and rewriting
  - CIL
  - 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

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
Abstraction

- Static analysis
  - abstraction is required for termination
    - Complete lattice
- 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) || (z->right == nilNode))
        y = z;
    else
        y = treeSuccessor(z->right);
    if (y->left != nilNode)      
        x = y->left;
    else
        x = y->right;
    x->parent = y->parent;
    if (y->parent == nilNode)     
        root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    if (y != z)                   
        z->key = y->key;
    return(y);
}
**Control Flow Paths**

*Static Analysis*

- All
- Feasible
- Executed

*Dynamic Analysis*

**Two Sides of Imprecision**

- **Imprecision in Dynamic Analysis**
  - \( \frac{\text{Feasible-Executed}}{\text{Feasible}} \)
  - increase precision as Executed approaches Feasible
    - systematic generation of tests

- **Imprecision in Static Analysis**
  - \( \frac{\text{All-Feasible}}{\text{All}} = \frac{\text{Infeasible}}{\text{Infeasible} + \text{Feasible}} \)
  - increase precision as Infeasible approaches 0
    - methods to eliminate infeasible paths
Interplay of Dynamic and Static Analysis

[Diagram showing interplay of dynamic and static analysis]

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?