

# Static and dynamic analysis: synergy and duality

Michael Ernst  
CSE 403, Spring 2018  
Lecture 2

# Static analysis

Examples: compiler optimizations, program verifiers

Examine program text (no execution)

Build a model of program state

- An abstraction of the run-time state

Reason over possible behaviors

- E.g., “run” the program over the abstract state

# Abstract interpretation

Typically implemented via dataflow analysis

Each program statement's *transfer function* indicates how it transforms state

Example: What is the transfer function for

**y = x++;**

?

# Selecting an abstract domain

$\langle x \text{ is odd}; y \text{ is odd} \rangle$

**$y = x++;$**

$\langle x \text{ is even}; y \text{ is odd} \rangle$

$\langle x \text{ is prime}; y \text{ is prime} \rangle$

**$y = x++;$**

$\langle x \text{ is anything}; y \text{ is prime} \rangle$

$\langle x = \{ 3, 5, 7 \}; y = \{ 9, 11, 13 \} \rangle$

**$y = x++;$**

$\langle x = \{ 4, 6, 8 \}; y = \{ 3, 5, 7 \} \rangle$

$\langle x=3, y=11 \rangle, \langle x=5, y=9 \rangle, \langle x=7, y=13 \rangle$

**$y = x++;$**

$\langle x=4, y=3 \rangle, \langle x=6, y=5 \rangle, \langle x=8, y=7 \rangle$

$\langle x_n = f(a_{n-1}, \dots, z_{n-1}); y_n = f(a_{n-1}, \dots, z_{n-1}) \rangle$

**$y = x++;$**

$\langle x_{n+1} = x_n + 1; y_{n+1} = x_n \rangle$

# Research challenge: Choose good abstractions

The abstraction determines the **expense** (in time and space)

The abstraction determines the **accuracy** (what information is lost)

- Less accurate results are poor for applications that require precision
- Cannot conclude all true properties in the grammar

# Static analysis recap

- **Slow** to analyze large models of state, so use abstraction
- **Conservative**: account for abstracted-away state
- **Sound**: (weak) properties are guaranteed to be true
  - \*Some static analyses are not sound

# Dynamic analysis

Examples: testing, profiling

Execute program (over some inputs)

- The compiler provides the semantics

Observe executions

- Requires instrumentation infrastructure

2 research challenges:

- what to measure
- what test runs

# Research challenge: What to measure?

Test oracle results

Coverage or frequency

- Statements, branches, paths, procedure calls, types, method dispatch

Values computed

- Parameters, array indices

Run time, memory usage

Similarities among runs [Podgurski 99, Reps 97]

Like abstraction, determines **what is reported**



# Research challenge: Choose good tests

The test suite determines the **expense** (in time and space)

The test suite determines the **accuracy** (what executions are never seen)

- Less accurate results are poor for applications that require correctness
- Many domains do not require correctness!

\*What information is being collected also matters

# Dynamic analysis recap

- Can be as **fast** as execution (over a test suite, and allowing for data collection)
  - Example: aliasing
- **Precise**: no abstraction or approximation
- **Unsound**: results may not generalize to future executions
  - Describes execution environment or test suite

# Static analysis

Abstract domain  
slow if precise

Conservative

due to abstraction

**Sound**

due to conservatism

# Dynamic analysis

Concrete execution  
slow if exhaustive

**Precise**

no approximation

Unsound

does not generalize

Use both!

# Same problem, different domain

Any analysis problem can be solved in either domain

- Type safety: no memory corruption or operations on wrong types of values
  - Static type-checking
  - Dynamic type-checking
- Slicing: what computations could affect a value
  - Static: reachability over dependence graph
  - Dynamic: tracing

# Memory checking

Goal: find array bound violations, uses of uninit. memory

Purify [Hastings 92]: run-time instrumentation

- Tagged memory: 2 bits (allocated, initialized) per byte
- Each instruction checks/updates the tags
  - Allocate: set “A” bit, clear “I” bit
  - Write: require “A” bit, set “I” bit
  - Read: require “I” bit
  - Deallocate: clear “A” bit

LCLint [Evans 96]: compile-time dataflow analysis

- Abstract state contains allocated and initialized bits
- Each transfer function checks/updates the state

Identical analyses!

Another example: atomicity checking [Flanagan 2003]

# Specifications

- Specification checking
  - Statically: theorem-proving
  - Dynamically: **assert** statement
- Specification generation
  - Statically: by hand or abstract interpretation [Cousot 77]
  - Dynamically: by invariant detection [Ernst 99], reporting unfalsified properties

# Static analysis

Abstract domain  
slow if precise

**Conservative**

due to abstraction

Sound

due to conservatism

# Dynamic analysis

Concrete execution  
slow if exhaustive

Precise

no approximation

**Unsound**

does not generalize

# Sound dynamic analysis

Observe every possible execution!

Problem: infinite number of executions

Solution: test case selection and generation

- Efficiency tweaks to an algorithm that works perfectly in theory but exhausts resources in practice



# Precise static analysis

Reason over full program state!

Problem: infinite number of executions

Solution: data or execution abstraction

- Efficiency tweaks to an algorithm that works perfectly in theory [Cousot 77] but exhausts resources in practice