Static Analysis

Martin Kellogg

Agenda: static analysis

- **high-level idea of static analysis**
- example static analysis: code review
- duality of static and dynamic analysis
- exam review (you ask questions)
- @7:30pm: exam

Motivation: many defects are hard to test for

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing
- Executing or dynamically analyzing all paths concretely to find such defects is **not feasible** (cf. exhaustive testing is infeasible)

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing
- Executing or dynamically analyzing all paths concretely to find such defects is **not feasible** (cf. exhaustive testing is infeasible)
- We want to learn about "**all possible runs**" of the program for particular properties

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing
- Executing or dynamically analyzing all paths concretely to find such defects is **not feasible** (cf. exhaustive testing is infeasible)
- We want to learn about "**all possible runs**" of the program for particular properties
  - Without actually running the program!

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing
- Executing or dynamically analyzing all paths concretely to find such defects is **not feasible** (cf. exhaustive testing is infeasible)
- We want to learn about "**all possible runs**" of the program for particular properties
  - Without actually running the program!
  - Bonus: we don't need test cases!

Motivation: many defects are hard to test for

- Many interesting defects are on **uncommon** or **difficult-to-exercise** execution paths
  - So it's hard to find them via testing
- Executing or dynamically analyz
  such defects is **not feasible** (cf.
- We want to learn about "**all po**
  particular properties
  - Without actually running t
  - Bonus: we don't need test ca

This is especially true for certain kinds of hard-to-test-for defects that might not be apparent even if you do exercise them, such as **resource leaks**

What does static analysis do well?

What does static analysis do well?

- Defects that result from inconsistently following **simple**, mechanical design **rules**

What does static analysis do well?

- Defects that result from inconsistently following **simple**, mechanical design **rules**
  - Security: buffer overruns, input validation
  - Memory safety: null pointers, initialized data
  - Resource leaks: memory, OS resources
  - API Protocols: device drivers, GUI frameworks
  - Exceptions: arithmetic, library, user-defined
  - Encapsulation: internal data, private functions
  - Data races: two threads, one variable

What does static analysis do well?

- Defects that result from inconsiste
  mechanical design **rules**
  - Security: buffer overruns, input
  - Memory safety: null pointers, in
  - Resource leaks: memory, OS resources
  - API Protocols: device drivers, GUI frameworks
  - Exceptions: arithmetic, library, user-defined
  - Encapsulation: internal data, private functions
  - Data races: two threads, one variable

There are **rules** for doing each of these things **correctly**, and a static analysis can automate those rules.

What is a static analysis?

What is a static analysis?

Definition: ***static analysis*** is the systematic examination of an abstraction of program state space

What is a static analysis?

Definition: ***static analysis*** is the systematic examination of an abstraction of program state space

- static analysis **does not execute** the program

What is a static analysis?

Definition: ***static analysis*** is the systematic examination of an abstraction of program state space

- static analysis **does not execute** the program
  - in contrast to a **dynamic analysis**, such as testing, which does execute the program

What is a static analysis?

Definition: ***static analysis*** is the systematic examination of an abstraction of program state space

- static analysis **does not execute** the program
  - in contrast to a **dynamic analysis**, such as testing, which does execute the program
- an **abstraction**, in this context, is a **selective representation** of the program that is simpler to analyze

What is a static analysis?

Definition: ***static analysis*** is the systematic examination of an abstraction of program state space

- static analysis **does not execute** the program
  - in contrast to a **dynamic analysis**, such as testing, which does execute the program
- an **abstraction**, in this context, is a **selective representation** of the program that is simpler to analyze
  - **key idea**: the abstraction will have fewer states to explore
    - hopefully, many fewer!

What is a static analysis?

Definition: ***static analysis*** is the sys
abstraction of program state space

This definition is most useful when thinking about **automated** static analyses. But whenever you reason through what a program does, you're doing static analysis by hand!

- static analysis **does not execut**
  - in contrast to a **dynamic anal**
    execute the program
- an **abstraction**, in this context, is a **selective representation** of the program that is simpler to analyze
  - **key idea**: the abstraction will have fewer states to explore
    - hopefully, many fewer!

Static analysis by hand: code review

Static analysis by hand: code review

Definition: In a ***code review***, another developer examines your proposed change and explanation, offers feedback, and decides whether to accept it.

Static analysis by hand: code review

Definition: In a ***code review***, another developer examines your proposed change and explanation, offers feedback, and decides whether to accept it.

- There is significant **tool support** for “modern” code review

Analogy: writing

Compare the effectiveness of:

- spell checking your own writing
- reading and editing your own writing
- having your writing be edited by someone else

Analogy: writing

Compare the effectiveness of:

- spell checking your own writing
- reading and editing your own writing
- having your writing be edited by someone else

Professional writers have **editors**; professional software engineers have **code reviewers**

What is(n't) "modern" code review?

- Historically, "code review" used to refer to what we now call ***code inspection*** or ***holistic code review***.

What is(n't) "modern" code review?

- Historically, "code review" used to refer to what we now call ***code inspection*** or ***holistic code review***.

Definition: a ***holistic code review*** is a code review of an entire component of a software system as a whole.

What is(n't) "modern" code review?

- Historically, "code review" used to refer to what we now call ***code inspection*** or ***holistic code review***.

Definition: a ***holistic code review*** is a code review of an entire component of a software system as a whole.

  - Typically, "code inspection" suggests that a team of reviewers is involved, while "holistic code review" suggests a single reviewer (but these are connotations, not rules)

So then what is modern code review?

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

Definition: a ***modern code review*** is a review of a set of proposed changes to a codebase, typically performed by another developer who is already familiar with the code being changed

So then what is modern code review?

- Unlike code inspections or holistic reviews, modern code reviews are performed at the **changeset** granularity

Definition: a ***modern code review*** is a review of a set of proposed changes to a codebase, typically performed by another developer who is already familiar with the code being changed

- Inductive argument for code quality:
  - if v(n) is good, and the diff between v(n) and v(n+1) is good, then v(n+1) is good

Aside: proof by induction

(on the whiteboard)

Modern code review: intuition

- “Given enough eyeballs, all bugs are shallow.” – Linus's Law

Modern code review: intuition

- “Given enough eyeballs, all bugs are shallow.” – Linus's Law
- Reviewer has:
    - different background, different experience
    - no preconceived idea of correctness
    - no bias because of “what was intended”

Modern code review: intuition

“Breadth of experience in an individual is essential to creativity and hence to good engineering. … Collective diversity, or diversity of the group - the kind of diversity that people usually talk about - is just as essential to good engineering as individual diversity. … Those **differences in experience are the "gene pool" from which creativity springs**.”

– Bill Wulf, National Academy of Engineering President

Modern code review: the most common analysis

- Modern code review is considered a **best practice almost everywhere** in industry

Modern code review: the most common analysis

"All code that gets submitted **needs to be reviewed** by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian to do code reviews, and obviously, **we spend a good chunk of our time reviewing code**."

- Amanda Camp, Software Engineer, Google

Modern code review: the most common analysis

“At Yelp we use review-board. An engineer works on a branch and commits the code to their own branch. The reviewer then goes through the diff, adds inline comments on review board and sends them back. The **reviews are meant to be a dialogue**, so typically comment threads result from the feedback. Once the reviewer's questions and concerns are all addressed they'll click "Ship It!" and the author will merge it with the main branch for deployment the same day.”

- Alan Fineberg, Software Engineer, Yelp

Modern code review: the most common analysis

“At Wizards we use Perforce for SCM. I work with stuff that manages rules and content, so we try to commit changes at the granularity of one bug at a time or one card at a time. Our team is small enough that you can designate one other person on team as a code reviewer. Usually you look at code sometime that week, but it depends on priority. **It’s impossible to write sufficient test harnesses** for the bulk of our game code, so **code reviews are absolutely critical**.”

- Jake Englund, Software Engineer, MtGO

Modern code review: the most common analysis

"At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, she submits it to this tool, which will notify the person or people she has asked to review the change, along with others that may be interested in the change – such as people who have worked on a function that got changed. At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."

Ryan McElroy, Software Engineer, Facebook

Modern code review: the most common analysis

- Modern code review is considered a **best practice almost everywhere** in industry
- While each place has their own way of doing reviews, the broad strokes are common between companies

Modern code review: benefits

Modern code review: benefits

- > 1 person has seen every piece of code
  - **Insurance** against author's disappearance (recall: bus factor)
  - **Accountability** (both author and reviewers are accountable)

Modern code review: benefits

- > 1 person has seen every piece of code
  - **Insurance** against author's disappearance (recall: bus factor)
  - **Accountability** (both author and reviewers are accountable)
- Forcing function for **documentation** and code improvements
  - Authors must articulate their decisions
  - Prospect of a review raises your quality threshold

Modern code review: benefits

- > 1 person has seen every piece of code
  - **Insurance** against author's disappearance (recall: bus factor)
  - **Accountability** (both author and reviewers are accountable)
- Forcing function for **documentation** and code improvements
  - Authors must articulate their decisions
  - Prospect of a review raises your quality threshold
- Inexperienced personnel get **experience** without hurting quality
  - Pairing them up with experienced developers
  - Can learn by being a reviewer as well

Modern code review: benefits

- > 1 person has seen every piece of code
  - **Insurance** against
  - **Accountability** (b
- Forcing function for
  - Authors must art
  - Prospect of a revi
- Inexperienced perso
  - Pairing them up wi
  - Can learn by being a reviewer as well

us factor)

untable)

ments

g quality

Non-goal: assessing whether the author is good at their job

- managers/HR **shouldn't** be involved in code review

Modern code review: benefits by the numbers

Modern code review: benefits by the numbers

- Average defect detection rates higher than testing
- 11 programs developed by the same group of people
  - First 5 without reviews: average 4.5 errors / 100 LoC
  - Remaining 6 with reviews: average 0.82 errors / 100 LoC
  - Errors reduced by > 80%.
- IBM's Orbit project: 500,000 lines, 11 levels of inspections. Delivered early with 1% of the predicted errors.
- After AT&T introduced reviews, 14% increase in productivity and a 90% decrease in defects.

(From Steve McConnell's Code Complete)

Code review: summary

- Modern code review is performed by **almost all** real software engineering teams (be worried if it's not!)
- Code review is the **single most common static analysis**
  - but it is not automated, so it's very expensive
- Code review is **very effective**
  - in some ways, even more effective than testing!

Agenda: static analysis

- high-level idea of static analysis
- example static analysis: code review
- **duality of static and dynamic analysis**
- exam review (you ask questions)
- @7:30pm: exam

Exam review: dynamic analysis

Exam review: dynamic analysis

- Execute program (over some inputs)
  - The compiler provides the semantics

Exam review: dynamic analysis

- Execute program (over some inputs)
  - The compiler provides the semantics
- Observe executions
  - Requires instrumentation infrastructure

Exam review: dynamic analysis

- Execute program (over some inputs)
  - The compiler provides the semantics
- Observe executions
  - Requires instrumentation infrastructure
- Analyze results

Exam review: dynamic analysis

- Execute program (over some inputs)
  - The compiler provides the semantics
- Observe executions
  - Requires instrumentation infrastructure
- Analyze results

These are the analyses that we've been studying this semester so far!

Exam review: dynamic analysis

- Execute program (over some inputs)
  - The **compiler provides the semantics**
- Observe executions
  - Requires instrumentation
- Analyze results

This means that we don't need an **external model** of what the computer does!

These are the analyses that we've been studying this semester so far!

Exam review: dynamic analysis properties

Exam review: dynamic analysis properties

- Can be as **fast** as execution (over a test suite, and allowing for data collection)
  - Example: aliasing

Exam review: dynamic analysis properties

- Can be as **fast** as execution (over a test suite, and allowing for data collection)
  - Example: aliasing
- **Precise**: no abstraction or approximation

Exam review: dynamic analysis properties

- 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 properties

Static analysis properties

- **Slow** to analyze large models of state, so use abstraction

Static analysis properties

- **Slow** to analyze large models of state, so use abstraction
- **Conservative**: account for abstracted-away state

Static analysis properties

- **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, but static analyses ***can*** be made sound

Static vs dynamic analyses

Dynamic analyses:

Static analyses:

Static vs dynamic analyses

Dynamic analyses:

- Concrete execution
  - slow if exhaustive

Static analyses:

- Abstract domain
  - slow if precise

Static vs dynamic analyses

Dynamic analyses:

- Concrete execution
  - slow if exhaustive
- **Precise**
  - no approximation

Static analyses:

- Abstract domain
  - slow if precise
- Conservative
  - due to abstraction

Static vs dynamic analyses

Dynamic analyses:

- Concrete execution
  - slow if exhaustive
- **Precise**
  - no approximation
- Unsound
  - does not generalize

Static analyses:

- Abstract domain
  - slow if precise
- Conservative
  - due to abstraction
- **Sound**
  - due to conservatism

Analogous analyses

- **Any** analysis problem can be solved with **either** a static or a dynamic analysis

Analogous analyses

- **Any** analysis problem can be solved with **either** a static or a dynamic analysis
  - e.g., consider **type safety**: no memory corruption or operations on wrong types of values

Analogous analyses

Aside: can you think of a language that doesn't have an analysis for type safety?

- **Any** analysis problem can be solved with **either** a static or a dynamic analysis
  - e.g., consider **type safety**: no memory corruption or operations on wrong types of values
    - Static type-checking (e.g., Java)
    - Dynamic type-checking (e.g., Python)

Static vs dynamic analyses

Dynamic analyses:

- Concrete execution
  - slow if exhaustive
- Precise
  - no approximation
- **Unsound**
  - does not generalize

Static analyses:

- Abstract domain
  - slow if precise
- **Conservative**
  - due to abstraction
- Sound
  - due to conservatism

Sound dynamic analysis?

Sound dynamic analysis?

- Observe **every possible execution**!

Sound dynamic analysis?

- Observe **every possible execution**!
- Problem: **infinite** number of executions

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?

Precise static analysis?

- Reason over **full program state**!

Precise static analysis?

- Reason over **full program state**!
- Problem: **infinite** number of executions

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 but exhausts resources in practice

Different subsets

- Dynamic analysis focuses on a **subset of executions**

Different subsets

- Dynamic analysis focuses on a **subset of executions**
  - i.e., the executions in the test suite, the executions that random input produces, etc.

Different subsets

- Dynamic analysis focuses on a **subset of executions**
  - i.e., the executions in the test suite, the executions that random input produces, etc.
  - typically **optimistic** about other executions
    - i.e., assume that they will be bug-free

Different subsets

- Dynamic analysis focuses on a **subset of executions**
  - i.e., the executions in the test suite, the executions that random input produces, etc.
  - typically **optimistic** about other executions
    - i.e., assume that they will be bug-free
- Static analysis focuses on a **subset of data structures**

Different subsets

- Dynamic analysis focuses on a **subset of executions**
  - i.e., the executions in the test suite, the executions that random input produces, etc.
  - typically **optimistic** about other executions
    - i.e., assume that they will be bug-free
- Static analysis focuses on a **subset of data structures**
  - more precise for data or control described by the abstraction

Different subsets

- Dynamic analysis focuses on a **subset of executions**
  - i.e., the executions in the test suite, the executions that random input produces, etc.
  - typically **optimistic** about other executions
    - i.e., assume that they will be bug-free
- Static analysis focuses on a **subset of data structures**
  - more precise for data or control described by the abstraction
  - typically **conservative** / **pessimistic** elsewhere
    - i.e., assume that unmodeled state is unsafe

Agenda: static analysis

- high-level idea of static analysis
- example static analysis: code review
- duality of static and dynamic analysis
- **exam review (you ask questions)**
- @7:30pm: exam