Static and dynamic analysis: synergy and duality

Michael Ernst
CSE 503, Winter 2018
Lecture 1
Static ❤ Dynamic Analysis

• Static and dynamic analyses are more similar than many people believe
• Encourage blending of the techniques and communities
• Productive research avenue
Outline

Review of static and dynamic analysis

Synergy: combining static and dynamic analysis
  • Aggregation
  • Analogies
  • Hybrids

Duality: subsets of behavior

Conclusion
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

\[ \langle x = \{ 3, 5, 7 \}; y = \{ 9, 11, 13 \} \rangle \]
\[ y = x++; \]
\[ \langle x = \{ 4, 6, 8 \}; y = \{ 3, 5, 7 \} \rangle \]

\[ \langle x=3, y=11 \rangle, \langle x=5, y=9 \rangle, \langle x=7, y=13 \rangle \]
\[ y = x++; \]
\[ \langle x=4, y=3 \rangle, \langle x=6, y=5 \rangle, \langle x=8, y=7 \rangle \]

\[ \langle x_n = f(a_{n-1}, \ldots, z_{n-1}); y_n = f(a_{n-1}, \ldots, z_{n-1}) \rangle \]
\[ y = x++; \]
\[ \langle x_{n+1} = x_n+1; y_{n+1} = x_n \rangle \]
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: profiling, testing

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?

Coverage or frequency
  • Statements, branches, paths, procedure calls, types, method dispatch

Values computed
  • Parameters, array indices

Run time, memory usage

Test oracle results

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
Outline

Review of static and dynamic analysis

⇒ Synergy: combining static and dynamic analysis
  • Aggregation
  • Analogies
  • Hybrids

Duality: subsets of behavior

Conclusion
Combining static and dynamic analysis

1. Aggregation:
   Pre- or post-processing

2. Inspiring analogous analyses:
   Same problem, different domain

3. Hybrid analyses:
   Blend both approaches
1. **Aggregation:**

   *Pre- or post-processing*

Use output of one analysis as input to another

**Dynamic then static**
- Profile-directed compilation: unroll loops, inline, reorder dispatch, …
- Verify properties observed at run time

**Static then dynamic**
- Reduce instrumentation requirements
  - Efficient branch/path profiling
  - Discharge obligations statically (type/array checks)
- Type checking (e.g., Java, including generics)
- Indicate suspicious code to test more thoroughly
2. Analogous analyses: 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
Your analogous analyses here

Look for gaps with no analogous analyses!
Try using the same analysis
  • But be open to completely different approaches
There is still low-hanging fruit to be harvested
3. Hybrid analyses: Blending static and dynamic analyses

Combine static and dynamic analyses

• Not mere aggregation, but a new analysis
• Disciplined trade-off between precision and soundness: find the sweet spot between them
Possible starting points

Analyses that trade off run-time and precision
- Different abstractions (at different program points)
- Switch between static and dynamic at analysis time

Ignore some available information
- Examine only some paths [Evans 94, Detlefs 98, Bush 00]

Merge based on observation that both examine only a subset of executions (next section of talk)
- Problem: optimistic vs. pessimistic treatment
- Fine-grained aggregation (concolic execution)

More examples: (bounded) model checking, security analyses, delta debugging [Zeller 99], etc.
Outline

Review of static and dynamic analysis
Synergy: combining static and dynamic analysis
  • Aggregation
  • Analogies
  • Hybrids
⇒ Duality: subsets of behavior
Conclusion
<table>
<thead>
<tr>
<th>Static analysis</th>
<th>Dynamic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract domain</td>
<td>Concrete execution</td>
</tr>
<tr>
<td>slow if precise</td>
<td>slow if exhaustive</td>
</tr>
<tr>
<td>Conservative</td>
<td>Precise</td>
</tr>
<tr>
<td>due to abstraction</td>
<td>no approximation</td>
</tr>
<tr>
<td>Sound</td>
<td>Unsound</td>
</tr>
<tr>
<td>due to conservatism</td>
<td>does not generalize</td>
</tr>
</tbody>
</table>
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
Dynamic analysis focuses on a subset of executions

The executions in the test suite

- Easy to enumerate
- Characterizes program use

Typically optimistic for other executions
Static analysis focuses on a subset of data structures

More precise for data or control described by the abstraction

- Concise logical description
- Typically conservative elsewhere (safety net)

Example: $k$-limiting [Jones 81]

- Represents each object reachable by $\leq k$ pointers
- Groups together (approximates) more distant objects
Dual views of subsets

Execution and data subsets are views on the same space. Every execution subset corresponds to a data subset:
- Executions induce data structures and control flow.

Every data subset corresponds to an execution subset:
- A set of objects represents the executions that generate them.

Subset description may be concise in one domain but complex in the other:
- What if the test suite was generated from a specification?

Any analysis may be conservative over other behaviors.
Differences between the approaches

Static and dynamic analysis communities work with different subsets

- Each subset and characterization is better for certain uses

What subsets have a concise description in both domains?

- Augment a test suite to fill out the data structures that it creates, making the data structure description a smaller logical formula
A hybrid view of subsets

Bring together static and dynamic analysis by unifying their subset descriptions

- Find subsets with small descriptions with respect to both data structures and executions
- Find a new, smaller description

Advantages of this approach

- Directly compare previous disparate analyses
- Directly apply analyses to other domain
- Switch between the approaches
- Obtain insight in order to devise and optimize analyses
Outline

Review of static and dynamic analysis
Synergy: combining static and dynamic analysis
  • Aggregation
  • Analogies
  • Hybrids
Duality: subsets of behavior
⇒ Conclusion
Potential pitfalls

Analogies between analyses
  • What applications tolerate unsoundness/imprecision?
  • Any more low-hanging fruit?
  • Most static and dynamic approaches differ

Hybrid analyses
  • How to measure and trade off precision and soundness
    • What is “partial soundness”? What is in between?
  • Not all static analyses are abstract interpretation
  • Optimistic vs. pessimistic treatment of unseen executions

Subset characterization
  • Find the unified characterization of behavior
Conclusion

Static and dynamic analysis share many similarities
  • Communities should be closer
Create analogous analyses
  • Many successes so far
Hybrid approach holds great promise
  • Analyses increasingly look like points in this continuum
  • Unified theory of subsets of executions/data is key

(Our) future work: explore this space
Discussion