Announcements

• “Last day”
• Presentations next week
  - Send me your title by tomorrow
  - Upload your slides at least 30 min before class
• Papers due May 6
• Extra credit opportunity: peer review of papers
  - Details forthcoming

Bor-Yuh Evan Chang - End-User Program Analysis for Data Structures

Software errors cost a lot

~$60 billion annually (~0.5% of US GDP)
  - 2002 National Institute of Standards and Technology report

> total annual revenue of Microsoft

> 10x annual budget of NST

Bor-Yuh Evan Chang - End-User Program Analysis for Data Structures

End-User Program Analysis for Data Structures

Bor-Yuh Evan Chang
University of Colorado
April 22, 2008

Collaborators: Xavier Rival (INRIA), George C. Necula (UC Berkeley)

But there’s hope in program analysis

Microsoft uses and distributes the Static Driver Verifier

Airbus applies the Astrée Static Analyzer

Companies, such as Coverity and Fortify, market static source code analysis tools

Bor-Yuh Evan Chang - End-User Program Analysis for Data Structures

Because program analysis can eliminate entire classes of bugs

For example,
  - Reading from a closed file: read( ); ☒
  - Reacquiring a locked lock: acquire( ); ☒

How?
  - Systematically examine the program
  - Simulate running program on “all inputs”
  - “Automated code review”

Bor-Yuh Evan Chang - End-User Program Analysis for Data Structures
Program analysis by example: Checking for double acquires

Simulate running program on “all inputs”

... code ...
// x now points to an unlocked lock
acquire(x);
... code ...

Must abstract

... code ...
// x now points to an unlocked lock in a linked list
acquire(x);
... code ...

To address the precision challenge

Traditional program analysis mentality:
“Why can’t developers write more specifications for our analysis? Then, we could verify so much more.”
“Since developers won’t write specifications, we will use default abstractions (perhaps coarse) that work hopefully most of the time.”

End-user approach:
“Can we design program analyses around the user? Developers write testing code. Can we adapt the analysis to use those as specifications?”

Overview of contributions
Extensible Inductive Shape Analysis
Precise inference of data structure properties
Able to check, for instance, the locking example
Targeted to software developers
Uses data structure checking code for guidance
- Turns testing code into a specification for static analysis
Efficient
- 10-100x speed-up over generic approaches
- Builds abstraction out of developer-supplied checking code

Summary of overview
Challenge in analysis: Finding a good abstraction
precise enough but not more than necessary
Powerful, generic abstractions
expensive, hard to use and understand
Built-in, default abstractions
often not precise enough (e.g., data structures)

End-user approach:
Must involve the user in abstraction
without expecting the user to be a program analysis expert
Shape analysis is a fundamental analysis

Data structures are at the core of
- Traditional languages (C, C++, Java)
- Emerging web scripting languages

Improves verifiers that try to
- Eliminate resource usage bugs (locks, file handles)
- Eliminate memory errors (leaks, dangling pointers)
- Eliminate concurrency errors (data races)
- Validate developer assertions

Enables program transformations
- Compile-time garbage collection
- Data structure refactorings

Shape analysis is not yet practical

Choosing the heap abstraction difficult for precision

Traditional approaches:
- Parametric in low-level, analyzer-oriented predicates
  - Very general and expressive
  - Hard for non-expert
- Built-in high-level predicates
  - Hard to extend
  - No additional user effort (if precise enough)

End-user approach:
- Parametric in high-level, developer-oriented predicates
  - Extensible
  - Targeted to developers

Key insight for being developer-friendly and efficient

Utilize “run-time checking code” as specification for static analysis.

Our framework is ...

An automated shape analysis with a precise memory abstraction based around invariant checkers.

- Extensible and targeted for developers
  - Parametric in developer-supplied checkers
- Precise yet compact abstraction for efficiency
  - Data structure-specific based on properties of interest to the developer
Shape analysis is an abstract interpretation on abstract memory descriptions with ...

Splitting of summaries

To reflect updates precisely

And summarizing for termination

Outline

Learn information about the checker to use it as an abstraction

1. splitting and interpreting update

2. type inference on checker definitions

Compare and contrast manual code review and our automated shape analysis

Overview: Split summaries to interpret updates precisely

Want abstract update to be “exact”, that is, to update one “concrete memory cell”.

The example at a high-level: iterate using cur: changing the doubly-linked list from purple to red.

“Split forward” by unfolding inductive definition

“Split backward” also possible and necessary

Technical Details:

How does the analysis do this unfolding? (Key: Segments are also inductively defined)

[POPL'08]

How does the analysis know to do this unfolding?

Contribution:

Turns testing code into specification for static analysis

How do we decide where to unfold?

Derives additional information to guide unfolding

How do we decide where to unfold?
Abstract memory as graphs

Make endpoints and segments explicit, yet high-level

Types for deciding where to unfold

Types make the analysis robust with respect to how checkers are written

Summary of checker parameter types

Summary of interpreting updates

Outline
Summary:
Given checkers, everything is automatic

Challenge:
Precision
(e.g., last, cur separated by at least one step)

Previous approaches guess where to fold for each graph.

Contribution:
Determine where by comparing graphs across history

Results: Performance
Times negligible for data structure operations (often in sec or 1/10 sec)

Benchmark | Max. Num. | Analysis Time (ms) |
------------|------------|-------------------|
Singly-linked list reverse | 1 | 1.0 |
Doubly-linked list remove | 5 | 17.9 |
Two-level skip list rebalance | 5 | 64.7 |
Linux kernel driver (894 loc) | 4 | 2969.6 |

Checked shape invariant as given by the checker is preserved across the operation.

Experience with the tool
Checkers are easy to write and try out
- Enlightening (e.g., red-black tree checker in 6 lines)
- Harder to “reverse engineer” for someone else’s code
- Default checkers based on types useful

Future expressiveness and usability improvements
- Pointer arithmetic and arrays
- More generic checkers:
  - polymorphic
  - “element kind unspecified”
  - higher-order parameterized by other predicates

Future evaluation: user study
Conclusion

Extensible Inductive Shape Analysis
precision demanding program analysis
improved by novel user interaction

Developer: 
Gets results corresponding to intuition

Analysis: 
Focused on what’s important to the developer

Practical precise tools for better software with an end-user approach!

What can inductive shape analysis do for you?

http://xisa.cs.berkeley.edu