Static Analysis in Practice

17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich

Quick Poll

- Who is familiar and comfortable with design patterns?
  - e.g. what is a Factory and why use it?

Outline: Static Analysis in Practice

- Case study: Analysis at eBay
- Case study: Analysis at Microsoft
- Analysis Results and Process
- Example: Standard Annotation Language
- Dataflow Analysis Soundness

Analyzing Tool Output

- True Positive – real issue
  - The error condition warned about can occur at run time, for some program input
- True Positive – but don't care
  - That program input is impossible due to the system context
    - e.g. security: attacker cannot control the input (are you sure?)
  - The error condition is not one we care about
- False Positive
  - The error condition cannot occur at run time, no matter what the program input is
  - Tool project: must distinguish from *true positive but don't care*
- False Negative
  - The program can get into an error condition for an attribute covered by the tool for some input, but the tool does not warn of it

Analysis Maturity Model

Caveat: not yet enough experience to make strong claims

- Level 1: use typed languages, ad-hoc tool use
- Level 2: run off-the-shelf tools as part of process
  - pick and choose analyses which are most useful
- Level 3: integrate tools into process
  - check in quality gates, milestone quality gates
  - integrate into build process, developer environments
  - use annotations/settings to teach tool about internal libraries
- Level 4: customized analyses for company domain
  - extend analysis tools to catch observed problems
- Level 5: continual optimization of analysis infrastructure
  - mine patterns in bug reports for new analyses
  - gather data on analysis effectiveness
  - tune analysis based on observations

Static Analysis in Engineering Practice

- A tool with different tradeoffs
  - Soundness: can find all errors in a given class
  - Focus: mechanically following design rules
- Major impact at Microsoft and eBay
  - Tuned to address company-specific problems
  - Affects every part of the engineering process

Outline: Static Analysis in Practice

- Case study: Analysis at eBay
- Case study: Analysis at Microsoft
- Analysis Results and Process
- Example: Standard Annotation Language
- Dataflow Analysis Soundness

Standard Annotation Language (SAL)

- A language for specifying contracts between functions
  - Intended to be lightweight and practical
  - More powerful—but less practical—contracts supported in systems like ESC/Java or Spec#
- Preconditions
  - Conditions that hold on entry to a function
  - What a function expects of its callers
- Postconditions
  - Conditions that hold on exiting a function
  - What a function promises to its callers
- Initial focus: memory usage
  - buffer sizes, null pointers, memory allocation…

SAL is checked using PREfast

- Lightweight analysis tool
  - Only finds bugs within a single procedure
  - Also checks SAL annotations for consistency with code
- To use it (for free!)
  - Download and install Microsoft Visual C++ 2005 Express Edition
    - http://msdn.microsoft.com/vstudio/express/visualc/
  - Download and install Microsoft Windows SDK for Vista
    - http://www.microsoft.com/downloads/details.aspx?familyid=c2b1e300-f358-4523-b479-f53d234cdccf
  - Use the SDK compiler in Visual C++
    - In Tools | Options | Projects and Solutions | VC++ Directories add C:\Program Files\Microsoft SDKs\Windows\v6.0\VC\Bin (or similar)
    - In project Properties | Configuration Properties | C/C++ | Command Line add /analyze as an additional option

Demonstration: PREfast

Buffer/Pointer Annotations

- Format
  - Leading underscore
  - List of components below

_in	The function reads from the buffer. The caller provides the buffer and initializes it.
_inout	The function both reads from and writes to buffer. The caller provides the buffer and initializes it.
_out	The function writes to the buffer. If used on the return value, the function provides the buffer and initializes it. Otherwise, the caller provides the buffer and the function initializes it.
_bcount(size)	The buffer size is in bytes.
_ecount(size)	The buffer size is in elements.
_opt	This parameter can be NULL.

PREfast: Immediate Checks

- Library function usage
  - deprecated functions
    - e.g. gets() vulnerable to buffer overruns
  - correct use of printf
    - e.g. does the format string match the parameter types?
  - result types
    - e.g. using macros to test HRESULTs
- Coding errors
  - = instead of == inside an if statement
- Local memory errors
  - Assuming malloc returns non-zero
  - Array out of bounds

Other Useful Annotations

__checkReturn	Callers must check the return value.

- .Net Annotation Format
  - Pre/Post attribute with arguments for the pre or postcondition
  - Surrounded in brackets
  - Alternative for the annotations above
  - Required for Tainted

[SA_Pre(Tainted=SA_Yes)]	This argument is tainted and cannot be trusted without validation.
[SA_Pre(Tainted=SA_No)]	This argument is not tainted and can be trusted
[SA_Post(Tainted=SA_No)]	Same as above, but useful as a postcondition

Other Supported Annotations

- How to test if this function succeeded
- How much of the buffer is initialized?
- Is a string null-terminated?
- Is an argument reserved?
- Is this an overriding method?
- Is this function a callback?
- Is this used as a format string?
- What resources might this function block on?
- Is this a fallthrough case in a switch?

SAL: the Benefit of Annotations

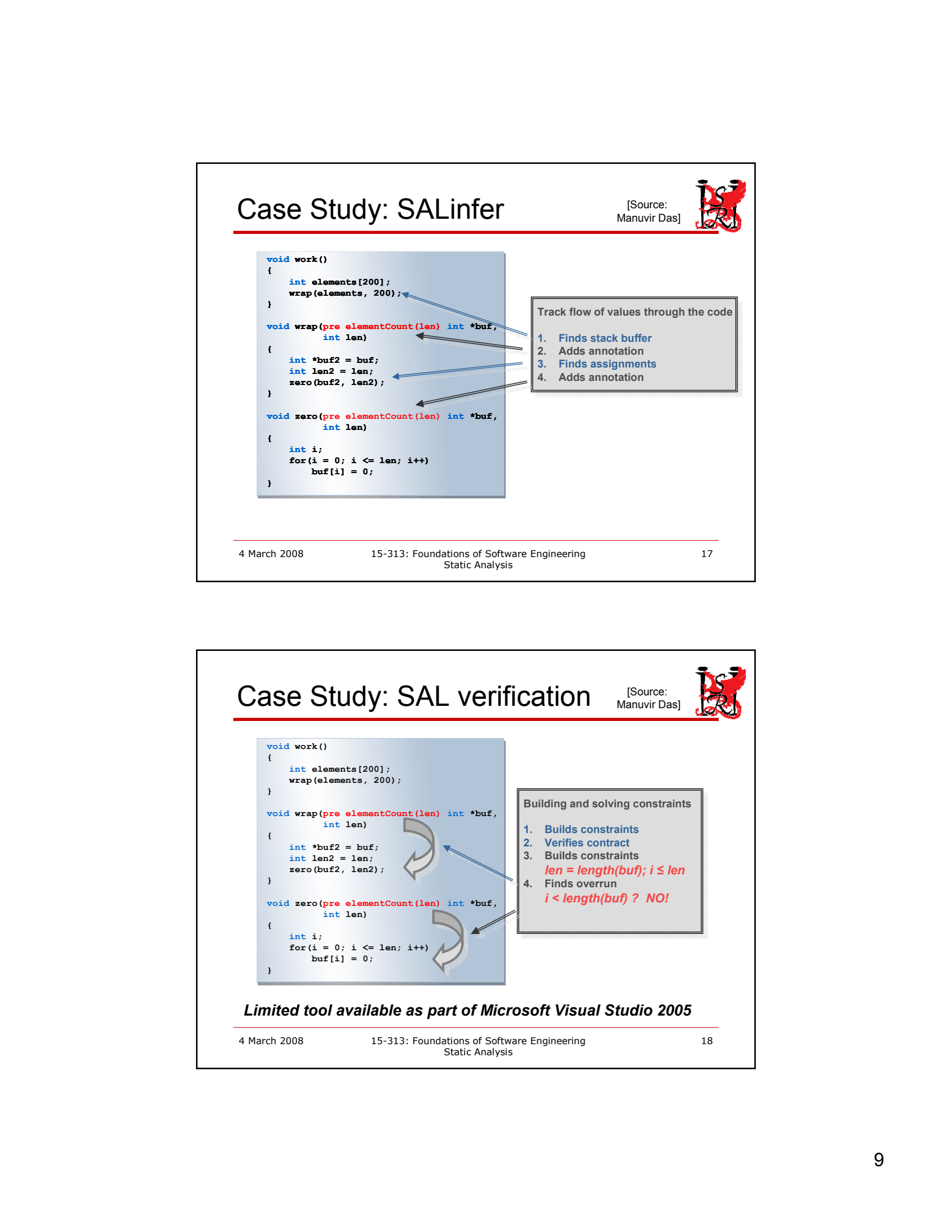
- Annotations express *design intent*
  - How you intended to achieve a particular quality attribute
    - e.g. never writing more than N elements to this array
- As you add more annotations, you find more errors
  - Get checking of library users for free
  - Plus, those errors are less likely to be false positives
    - The analysis doesn't have to guess your intention
- Annotations also improve scalability
  - PreFAST uses very sophisticated analysis techniques
  - These techniques can't be run on large programs
  - Annotations isolate functions so they can be analyzed one at a time

SAL: the Benefit of Annotations

- How to motivate developers?
  - Especially for millions of lines of unannotated code?
- Microsoft approach
  - Require annotations at checkin
    - Reject code that has a char* with no __ecount()
  - Make annotations natural
    - Ideally what you would put in a comment anyway
      - But now machine checkable
      - Avoid formality with poor match to engineering practices
  - Incrementality
    - Check code ↔ design consistency on every compile
    - Rewards programmers for each increment of effort
      - Provide benefit for annotating partial code
      - Can focus on most important parts of the code first
      - Avoid excuse: I'll do it after the deadline
  - Build tools to infer annotations
    - Inference is approximate
      - May need to change annotations
      - Hopefully saves work overall
    - Unfortunately not yet available outside Microsoft

Case Study: SALinfer

[Source: Manuvir Das]



```
void work()
{
    int elements[200];
    wrap(elements, 200);
}

void wrap(pre elementCount(len) int *buf,
          int len)
{
    int *buf2 = buf;
    int len2 = len;
    zero(buf2, len2);
}

void zero(pre elementCount(len) int *buf,
          int len)
{
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;
}
```


Case Study: SAL verification

[Source: Manuvir Das]


```
void work()
{
    int elements[200];
    wrap(elements, 200);
}

void wrap(pre elementCount(len) int *buf,
          int len)
{
    int *buf2 = buf;
    int len2 = len;
    zero(buf2, len2);
}

void zero(pre elementCount(len) int *buf,
          int len)
{
    int i;
    for(i = 0; i <= len; i++)
        buf[i] = 0;
}
```


Limited tool available as part of Microsoft Visual Studio 2005

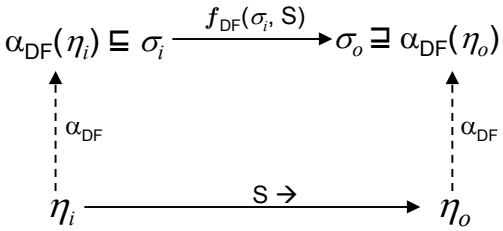
Outline: Static Analysis in Practice

- Case study: Analysis at eBay
- Case study: Analysis at Microsoft
- Analysis Results and Process
- Example: Standard Annotation Language
- Dataflow Analysis Soundness

Dataflow Analysis Soundness

- Intuition
  - The result of dataflow analysis is a conservative approximation of all possible run time states
- Definition
  - A dataflow analysis is sound if, for all programs P, for all inputs I, for all times T in P's execution on input I,
  - If P is at statement S at time T, with memory η, and the analysis returned abstract state σ for S,
  - then $\alpha(\eta) \sqsubseteq \sigma$

Local Soundness

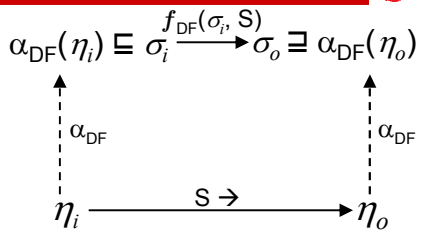


- Local correctness condition for dataflow analysis
  - If applying a transfer function to statement S and input information σ_i yields output information σ_o,
  - Then for all program states η_i such that $\alpha(\eta_i) \sqsubseteq \sigma_i$ and executing S in state η_i yields state η_o,
  - We must have $\alpha(\eta_o) \sqsubseteq \sigma_o$
- Global soundness follows from local soundness by induction
  - Initial dataflow facts are sound
  - Each step in program execution maintains soundness invariant

Finding Errors with Local Soundness

- Consider the **incorrect** flow function:

$f_{ZA}(\sigma, [x := y \ op \ z]) =$
if $\sigma[y]$=Z || $\sigma[z]$=Z
then $[x \mapsto Z]\sigma$
else $[x \mapsto MZ]\sigma$



- Local Soundness fails!
  - Let σ_i = [], S = "x := 3+0"
  - Consider η_i = []. As required, $\alpha_{DF}(\eta_i)$ = [] $\sqsubseteq \sigma_i$
  - Now $\sigma_o = f_{DF}(\sigma_i, S)$ = [x↦Z]
  - And η_o = S(η_i) = [x↦3]
  - So $\alpha_{DF}(\eta_o) = \alpha_{DF}([x \mapsto 3])$ = [x↦NZ]
  - BUT $\alpha_{DF}(\eta_o) \not\sqsubseteq \sigma_o$ because Z $\not\sqsubseteq$ NZ, so local soundness is violated

Why care about Soundness?

- Analysis Producers
  - Writing analyses is hard
    - People make mistakes all the time
    - Need to know how to ***think*** about correctness
    - When the analysis gets tricky, it's useful to prove it correct formally
- Analysis Consumers
  - Sound analysis provides guarantees
    - Optimizations won't break the program
    - Finds all defects of a certain sort
  - Decision making
    - Knowledge of soundness techniques lets you ask the right questions about a tool you are considering
    - Soundness affects where you invest QA resources
      - Focus testing efforts on areas where you don't have a sound analysis!

Questions?