The Bugs and the Bees

Research in Programming Languages and Security

David Evans
evans@cs.virginia.edu
http://www.cs.virginia.edu/~evans


University of Virginia
Department of Computer Science

What is Computer Science?

Let *AB* and *CD* be the two given numbers not relatively prime. It is required to find the greatest common measure of *AB* and *CD*.

If now *CD* measures *AB*, since it also measures itself, then *CD* is a common measure of *CD* and *AB*. And it is manifest that it is also the greatest, for no greater number than *CD* measures *CD*. But, if *CD* does not measure *AB*, then, when the less of the numbers *AB* and *CD* being continually subtracted from the greater, some number is left which measures the one before it.

For a unit is not left, otherwise *AB* and *CD* would be relatively prime, which is contrary to the hypothesis. Therefore some number is left which measures the one before it. Now let *CD*, measuring *BE*, leave *EA* less than itself, let *EA*, measuring *DF*, leave *FC* less than itself, and let *CF* measure *AE*.

Since then, *CF* measures *AE*, and *AE* measures *DF*, therefore *CF* also measures *DF*. But it measures itself, therefore it also measures the whole *CD*. But *CD* measures *BE*, therefore *CF* also measures *BE*. And it also measures *EA*, therefore it measures the whole *BA*. But it also measures *CD*, therefore *CF* measures *AB* and *CD*. Therefore *CF* is a common measure of *AB* and *CD*.

I say next that it is also the greatest. If *CF* is not the greatest common measure of *AB* and *CD*, then some number *G*, which is greater than *CF*, measures the numbers *AB* and *CD*.

Now, since *G* measures *CD*, and *CD* measures *BE*, therefore *G* also measures *BE*. But it also measures the whole *BA*, therefore it measures the remainder *AE*. But *AE* measures *DF*, therefore *G* also measures *DF*. And it measures the whole *DC*, therefore it also measures the remainder *CF*, that is, the greater measures the less, which is impossible. Therefore no number which is greater than *CF* measures the numbers *AB* and *CD*. Therefore *CF* is the greatest common measure of *AB* and *CD*.

Euclid's Elements, Book VII, Proposition 2 (300BC)

By the word operation, we mean any process which alters the mutual relation of two or more things, be this relation of what kind it may. This is the most general definition, and would include all subjects in the universe. Again, it might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine... Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

Ada, Countess of Lovelace, around 1830

What is the difference between Euclid and Ada?

"It depends on what your definition of 'is' is."
Bill Gates
(speaking at Microsoft's anti-trust trial)

Geometry vs. Computer Science

- Geometry (mathematics) is about *declarative* knowledge: "what is"

  If now *CD* measures *AB*, since it also measures itself, then *CD* ***is*** a common measure of *CD* and *AB*
- Computer Science is about *imperative* knowledge: "how to"

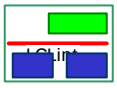
Computer Science has **nothing** to do with beige (or translucent blue) boxes called "computers" and is not a science.

Computer Science

- "How to" knowledge:
  - Ways of describing imperative processes (computations)
  - Ways of reasoning about (predicting) what imperative processes will do
- Most interesting CS problems concern:
  - Better ways of describing computations
  - Ways of reasoning about what they do (and don't do)

My Research Projects

- The Bugs

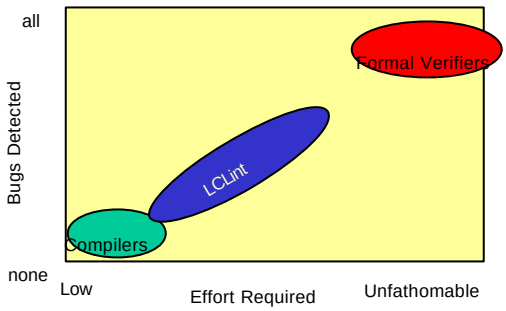


How can we detect code that describes unintended computations?

- The Bees - "Programming the Swarm"

How can we program large collections of devices and reason about their behavior?

A Gross Oversimplification



Everyone Likes Types

- Easy to Understand
- Easy to Use
- Quickly Detect Many Programming Errors
- Useful Documentation
- **…even though they are lots of work!**
  - 1/4 of text of typical C program is for types

Limitations of Standard Types

Type of reference never changes

Language defines checking rules

One type per reference

Limitations of Standard Types

	Attributes
Type of reference never changes	State changes along program paths
Language defines checking rules	System *or programmer* defines checking rules
One type per reference	Many attributes per reference

Approach

- Programmers add annotations (formal specifications)
  - Simple and precise
  - Describe programmers intent:
    - Types, memory management, data hiding, aliasing, modification, null-ity, buffer sizes, security, etc.
- LCLint detects inconsistencies between annotations and code
  - Simple (fast!) dataflow analyses

Example: Buffer Overflows

- [David Larochelle's MCS]
- Most commonly exploited security vulnerability
  - 1988 Internet Worm
  - Still the most common attack
    - Code Red exploited buffer overflow in IIS
    - >50% of CERT advisories, 23% of CVE entries in 2001
- Attributes describe sizes of allocated buffers

Buffer Overflow Example

Source: Secure Programming, SecurityFocus.com

```
char *strncat (char *s1, char *s2, size_t n)
 /*@requires maxSet(s1) >=maxRead(s1) + n@*/

void func(char *str) {
  char buffer[256];
  strncat(buffer, str, sizeof(buffer) - 1); }
```

uninitialized array

```
strncat.c:4:21: Possible out-of-bounds store:
    strncat(buffer, str, sizeof((buffer)) - 1);
 Unable to resolve constraint:
  requires maxRead (buffer @ strncat.c:4:29) <= 0
 needed to satisfy precondition:
  requires maxSet (buffer @ strncat.c:4:29)
           >= maxRead (buffer @ strncat.c:4:29) + 255
  ...
```


Detecting Buffer Overflows

- Annotations express constraints on buffer sizes
  - e.g., maxSet is the highest index that can safely be written to
- Checking uses axiomatic semantics with simplification rules
- Heuristics for analyzing common loop idioms
- Detected known and unknown vulnerabilities in w u-ftpd and BIND

LCLint Status

- Public distribution
- Effective checking >100K line programs (checks about 1K lines per second)
  - Detects lots of real bugs in real programs (including itself, of course)
  - Real users, C Unleashed, Linux Journal, etc.
- Checks include type abstractions, modifications, globals, memory leaks, dead storage, naming conventions, undefined behavior, incomplete definition...

Some Open Issues

- Integrate run-time checking
  - Combine static and run-time checking to enable additional checking and completeness guarantees
- Generalize framework
  - Support static checking for multiple source languages in a principled way
- Design-level Properties
- Concurrent programs
- Make it easier to annotate legacy programs

LCLint

- More information: lclint.cs.virginia.edu
  USENIX Security '01, PATV '2000, PLDI '96
- Public release – real users, mentioned in C FAQ, C Unleashed, Linux Journal, etc.
- Students (includes other PL/SE/security related projects):
  - David Larochelle: buffer overflows, automatic annotations
  - Joel Winstead: parallel loop exception semantics
  - Greg Yukl: serialization
  - Undergraduates: David Friedman, Mike Lanouette, Lim Lam, Tran Nguyen, Hien Phan, Adam Sowers
- Current Funding: NASA (joint with John Knight)

Programming the Swarm

Really Brief History of Computer Science

1950s: Programming in the small...
- Programmable computers
- Learned the programming is hard
- Birth of higher-order languages
- Tools for reasoning about trivial programs

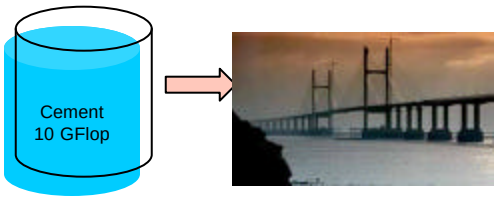
1970s: Programming in the large...
- Abstraction, objects
- Methodologies for development
- Tools for reasoning about component-based systems

2000s: Programming the Swarm!

What's Changing

- Execution Platforms
  - Not computers (98% of microprocessors sold this year)
  - Small and cheap
- Execution environment
  - Interact with physical world
  - Unpredictable, dynamic
- Programs
  - Old style of programming won't work
  - Is there a new paradigm?

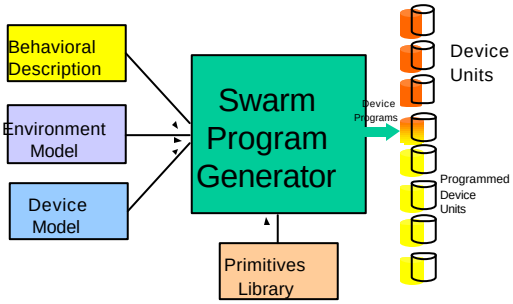
Programming the Swarm: Long-Range Goal



Why this Might be Possible?

- Biology Does It
  - Ant routing
    - Find best route to food source using pheromone trails
  - Bee house-hunting
    - Reach consensus by dancing and split to new hive
  - Complex creatures self-organize from short DNA program and dumb chemicals
    - Genetic code for 2 humans differs in only 2M base pairs (.5 MB < 1% of Win2000)

Swarm Programming Model



Swarm Programming

- Primitives describe group behaviors
  - What are the primitives?
  - How are they specified?
    - Important to understand both functional (how the state changes) and non-functional (power use, robustness, efficiency, etc.) properties
- Construct complex behaviors by composing primitives
  - What are the right combination mechanisms?
  - Pick the right combination of primitive implementations based on description of desired non-functional properties

Open Issues

- How can we predict the functional and non-functional properties of combinations of primitives?
- How can we synthesize efficient swarm programs from a library of primitive implementations?
- Security
  - Can we use swarm programming to build systems that are resilient to classes of attack?
  - Can we produce swarm programs with known behavioral constraints?

Programming the Swarm

swarm.cs.virginia.edu

- Students:
  - Gilbert Backers: Adaptive Hierarchal Communication
  - Weilin Zhong: Security of Ant Routing
  - Undergraduates: Keen Browne, Mike Cuvelier, John Calandrino, Bill Oliver, Mike Hoyge, Jon McCune, Errol McEachron
- Funding: NSF Career Award

Choosing an Advisor

- **Most important decision you will make in graduate school!**
- **Don't rely on the matching process**
  - This is a LAST RESORT
  - If you don't know who your advisor is before the matching process, something is wrong

Things you should do:

- Talk to faculty – don't wait until the week before matching forms are due!
- Talk to students about their advisors
- Think of your own project ideas
- Prove your value as a student to a potential advisor
- But also – expect potential advisor to demonstrate their value as an advisor

Summary

- Computer Science is about "how to" knowledge
- Interesting problems:
  - Describing and reasoning about behavior of large ad hoc collections (Programming the Swarm)
  - Detecting differences between what programs express and what programmers intend (LCLint)
- Be proactive about finding an advisor
- [Swarm Demo]
- evans@cs.virginia.edu