Weird things that surprise academics trying to commercialize a static checking tool.

Andy Chou, Ben Chelf, Seth Hallem
Charles Henri-Gros, Bryan Fulton, Ted Unangst
Chris Zok
Coventry
Dawson Engler
Stanford

Myth: the C (or C++) language exists.

Well, not really. The standard is not a compiler.

Rule: static analysis must compile code to check. If you cannot (correctly) parse “language” cannot check.

Common (mis)usage model: “allegedly C” header file does something bizarre not-C thing. Included by all source. Customer watches your compiler emit voluminous parse errors. (This is not impressive.) Of course: gets way worse with C++ (which we support)

A naïve view

- Initial market analysis: "We handle linux, bsd, we just need a pretty box!" Not quite.
- First rule of static analysis: no check, no bug. Two first order examples we never would have guessed. Problem 1: if you can't find the code, can't check it. Problem 2: if you can't compile code, you can't check it.

Some bad examples to find in headers

- Banal. But take more time than you can believe:

```c
void x;
short x; int *y = d(int);c;
int foo(int a, int a);
unsigned x @ "TEXT": // unless ”-packed"
  _packed (...) struct foo { (...) };

unsigned x = Oxdead_bee;

End lines with "\r" rather than "\n"

And, of course, asm:

// newline = end
asm mov eax, eab
asm foo();
asm (mov eax, eab;);
asm_end_asm;

// "J" = end
asm mov eax, eab
```

Microsoft example: precompiled headers

- Spec:
The compiler treats all code occurring before the .h file as precompiled. It skips to just beyond the #include directive associated with the .h file, uses the code contained in the .ph file, and then compiles all code after filename

- Implication
I can put whatever I want here. It doesn't have to compile. If your compiler gives an error it sucks. #include <some-precompiled-header.h>

- It gets worse: on-the-fly header fabrication

Solution: pre-preprocessing rewrite rules.

- Supply regular expressions to rewrite bad constructs

```c
#pragma asm
#pragma end_asm

ppp_translate("/\#pragma asm/\#if 0/");
ppp_translate("/\#pragma end_asm/\#endif/");

#pragma asm
#pragma end_asm

#endif
```
What this all means concretely.

- We use Edison Design Group (EDG) frontend
  Aggressive support for gcc, microsoft, etc. (bug compat)
- Still: coverage by far the largest source of EDG bugs:
  146 parsing test cases (i.e., we got burned)
  219 compiler line translation test cases (i.e., ibid).
  163 places where frontend hacked ("#ifdef COVERITY")
- Still need custom rewriter for many supported compilers:

```
205 hpux_compilers.c  483 sun_compilers.c
215 iar_compiler.c    485 arm_compilers.c
240 ti_compiler.c     617-gnu_compilers.c
251 green_hills_compiler.c 746 microsoft_compilers.c
377 intel_compilers.c 1587 metroworks_compilers.c
```

Some experience.

- Surprise: Sales guys are great
  Easy to evaluate. Modular.
- Company X buys tool, then downsizes. Good or bad?
  For sales, very good: X fires 110 people. They get jobs
  elsewhere. Recommend coverage. 4 closed deals.
- Large companies "want" to be honest
  Veritas: want monitoring so don't accidently violate!
- What can you sell?
  Makes it difficult to deploy anything sophisticated.
  Example: statistical inference.
  Some ways, checkers lag much behind our research ones.

Coverity's commercial history

<table>
<thead>
<tr>
<th>Breakthrough technology out of Stanford</th>
<th>2002</th>
<th>2003</th>
<th>2004-05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company incorporated</td>
<td>2002</td>
<td>2003</td>
<td>2004-05</td>
</tr>
<tr>
<td>Achieved profitability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product growth and proliferation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Meta-level compilation checker
  ("Stanford Checker") detects 2000+ bugs in Linux.
- "We'll just charge per seat like everyone else"
  Finish the story: "Company X buys three Purify seats, one for
  Asian, one for Europe and one for the US..."
- Try #2: "We'll charge per lines of code"
  "That is a really stupid idea: (1) ... (2) ...
  Actually works. I'm still in shock. Would recommend it.
- Good feature for seller:
  No seat games. Revenue grows with code size. Run on
  another code base = new sale.
- Good feature for buyer: No seat-model problems
  Buy once for project, then done. No per-seat or per-
  usage cost; no node lock problems; no problems adding,
  removing or renaming developers (or machines)
  People actually seem to like this pitch.

"No, your tool is broken: that’s not a bug"

- "No, the loop will go through once!"
  ```
  for(i=0; i < n; i++) {
    printf('dead code...
    }
  ```
- "No, && is ‘or’!
  ```
  void foo(void * p, void * q) {
    if(p && q) return 0;
  ```
- "No, ANSI lets you write I past end of the array!"
  ("We’ll have to agree to disagree.
  ```
  unsigned p[4]; p[4] = 1;
  ```

A partial list of 70+ customers...

- EDA
  - Synopsys
    - Cadence
      - Checkpoint
  - Storage
    - Veritas
  - Security
    - NukeBox
    - CheckPoint
  - Networking
    - BlueSilver
  - Government
    - Oracle
  - Embedded
    - Wind River
  - Open Source
    - Oracle
    - Apache
Slide 8

DE6 whole bunch of options: razor blade model, where we give away checkers for free and charge for system. or inverse razor where we give away system and charge for checkers. or charge per seat, or charge per lines of code (prefix). get a lot of pushback on the last one. prefix worked ok, but not what we would consider a success.

i argued very strongly against per line model. completely wrong.
Dawson Engler, 8/23/2005

Slide 9

DE7 count how often something true versus not. sort in decreasing deviance. inspect until hit fp. developer inspect all, mark as FP. say tool sucks to everyone.
Dawson Engler, 8/23/2005

Slide 10

DE8 we know about these since happen in results meetings. a bit dangerous, but usually other developers will laugh at the confused one. scary to think of other times when things just marked as FP.
Dawson Engler, 8/23/2005
**Summary**

- Static analysis
  Better at checking surface properties
  Big wins: don’t run code, all paths. Easy diagnosis.
  Low incremental cost per line of code
  Can get results in an afternoon: much easier to commercialize.
  10-100x more bugs.
- Model checking
  Better at checking code implications.
  Major win over testing: explore all actions a state can do
  before going to next
  Makes low-probability events as probable as high.
  Works very well when massive interleavings and bugs
  horrible.

**Open Q: how to get the bugs that matter?**

- Myth: all bugs matter and all will be fixed
  *FALSE*
  Find 10 bugs, all get fixed. Find 10,000...
- Reality
  All sites have many open bugs (observed by us & PREfix)
  Myth lives because state-of-art is so bad at bug finding
  What users really want: The 5-10 that "really matter"
- General belief: bugs follow 90/10 distribution
  Out of 1000, 100 account for most pain.
  Fixing 900 waste of resources & may make things worse
- How to find worst? No one has a good answer to this.

**Open Q: Do static tools really help?**

**Some cursory static analysis experiences**

- Bugs are everywhere
  Initially worried we’d resort to historical data...
  100 checks? You’ll find bugs (if not, bug in analysis)
- Finding errors often easy, saying why is hard
  Have to track and articulate all reasons.
- Ease-of-inspection “crucial”
  Extreme: Don’t report errors that are too hard.
- The advantage of checking human-level operations
  Easy for people? Easy for analysis. Hard for analysis?
  Hard for people.
- Soundness not needed for good results.

**Myth: more analysis is always better**

- Does not always improve results, and can make worse
- The best error:
  Easy to diagnose
  True error
- More analysis used, the worse it is for both
  More analysis = the harder error is to reason about,
  since user has to manually emulate each analysis step.
  Number of steps increase, so does the chance that one
  went wrong. No analysis = no mistake.
- In practice:
  Demote errors based on how much analysis required
  Revert to weaker analysis to cherry pick easy bugs
  Give up on errors that are too hard to diagnose.

**Myth: Soundness is a virtue.**

- Soundness: Find all bugs of type X.
  Not a bad thing. More bugs good.
  BUT: can only do if you check weak properties.
- What soundness really wants to be when it grows up:
  Total correctness: Find all bugs.
  Most direct approximation: find as many bugs as possible.
- Opportunity cost:
  Diminishing returns: Initial analysis finds most bugs
  Spend on what gets the next biggest set of bugs
  Easy experiment: bug counts for sound vs unsound tools.
- End-to-end argument:
  "It generally does not make much sense to reduce the
  residual error rate of one system component (property)
  much below that of the others."