Valgrind

A Framework for Heavyweight Dynamic Binary Instrumentation

Nicholas Nethercote – National ICT Australia

Julian Seward – OpenWorks LLP

FAQ #1

- How do you pronounce “Valgrind”?
- “**Val-grinned**”, not “Val-grined”
- Don’t feel bad: almost everyone gets it wrong at first

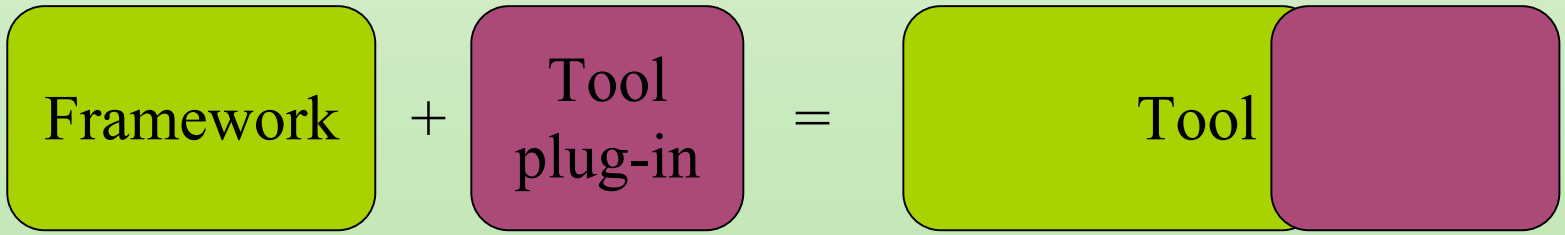
DBA tools

- Program analysis tools are useful
  - Bug detectors
  - Profilers
  - Visualizers

- **Dynamic binary analysis** (DBA) tools
  - Analyse a program's machine code at run-time
  - Augment original code with **analysis code**

Building DBA tools

- **Dynamic binary instrumentation** (DBI)
  - Add analysis code to the original machine code at run-time
  - No preparation, 100% coverage
- DBI frameworks
  - Pin, DynamoRIO, Valgrind, etc.



Prior work

Well-studied	Not well-studied
Framework performance	Instrumentation capabilities
Simple tools	Complex tools

- **Potential of DBI has not been fully exploited**
  - Tools get less attention than frameworks
  - Complex tools are more interesting than simple tools

Shadow value tools

Shadow value tools (I)

- Shadow every value with another value that describes it
  - Tool stores and propagates shadow values in parallel

	Tool(s)	Shadow values help find…
bugs	**Memcheck**	**Uses of undefined values**
	Annelid	Array bounds violations
	Hobbes	Run-time type errors
security	TaintCheck, LIFT, TaintTrace	Uses of untrusted values
	"Secret tracker"	Leaked secrets
properties	DynCompB	Invariants
	Redux	Dynamic dataflow graphs

Memcheck

- Shadow values: defined or undefined

Original operation	Shadow operation
int* p = malloc(4)	sh(p) = undefined
R1 = 0x12345678	sh(R1) = defined
R1 = R2	sh(R1) = sh(R2)
R1 = R2 + R3	sh(R1) = add_{sh}(R2, R3)
if R1==0 then goto L	complain if sh(R1) is undefined

- 30 undefined value bugs found in OpenOffice

Shadow value tools (II)

- All shadow value tools work in the same basic way
- Shadow value tools are **heavyweight** tools
  - Tool's data + ops are as complex as the original programs's
- Shadow value tools are hard to implement
  - Multiplex real and shadow registers onto register file
  - Squeeze real and shadow memory into address space
  - Instrument most instructions and system calls

Valgrind basics

Valgrind

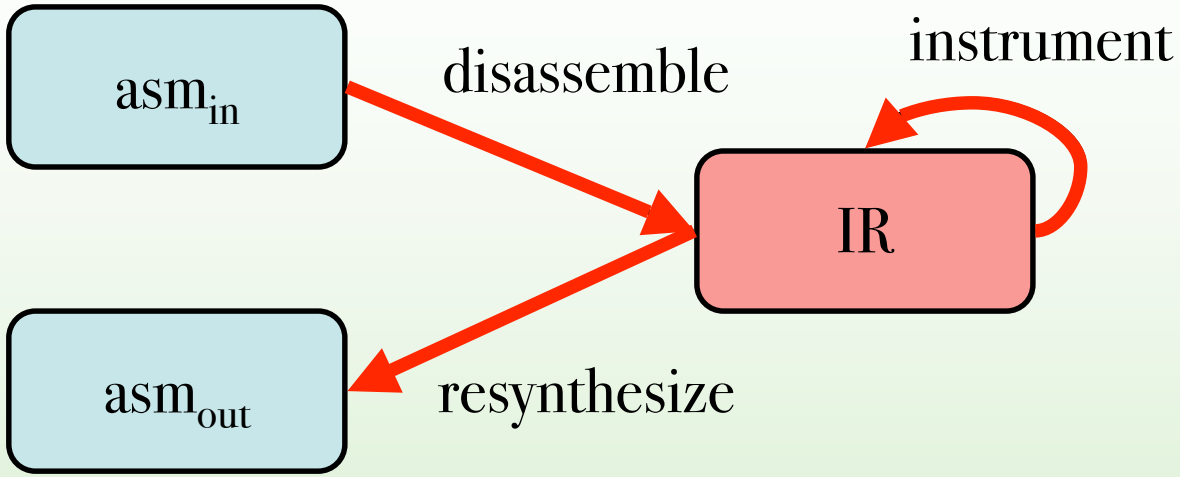
- Software
  - Free software (GPL)
  - {x86, x86-64, PPC}/Linux, PPC/AIX
- Users
  - Development: Firefox, OpenOffice, KDE, GNOME, MySQL, Perl, Python, PHP, Samba, RenderMan, Unreal Tournament, NASA, CERN
  - Research: Cambridge, MIT, Berkeley, CMU, Cornell, UNM, ANU, Melbourne, TU Muenchen, TU Graz
- Design
  - Heavyweight tools are well supported
  - Lightweight tools are slow

Two unusual features of Valgrind

#1: Code representation

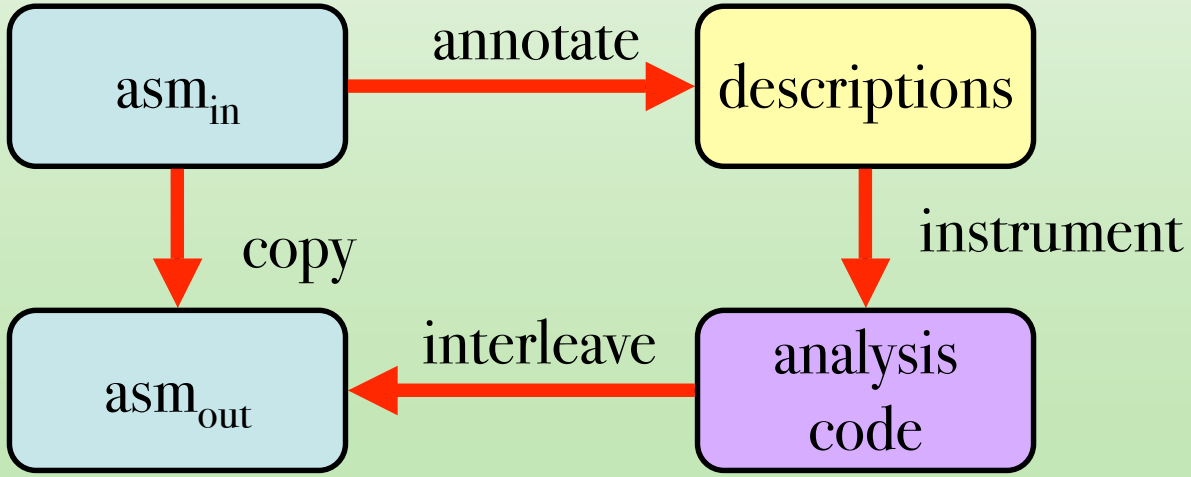
D&R

Disassemble-and-resynthesize (Valgrind)



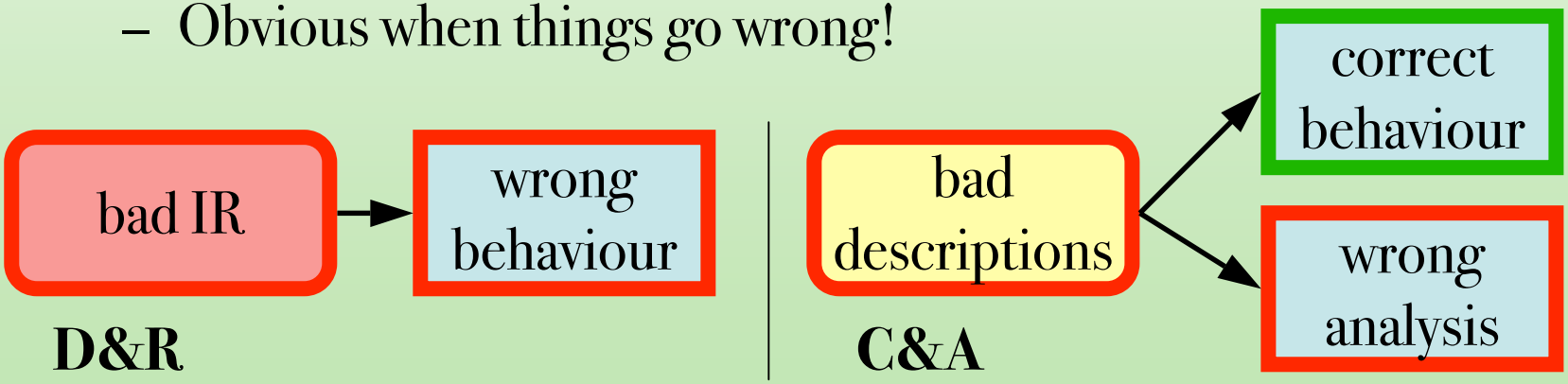
C&A

Copy-and-annotate



Pros and cons of D&R

- Cons: Lightweight tools
  - Framework design and implementation effort
  - Code translation cost, code quality
- Pros: Heavyweight tools
  - Analysis code as expressive as original code
  - Tight interleaving of original code and analysis code
  - Obvious when things go wrong!



Other IR features

Feature	Benefit
First-class shadow registers	As expressive as normal registers
Typed, SSA	Catches instrumentation errors
RISC-like	Fewer cases to handle
Infinitely many temporaries	Never have to find a spare register

- Writing complex inline analysis code is easy

#2: Thread serialisation

- Shadow memory: memory accesses no longer atomic
  - Uni-processors: thread switches may intervene
  - Multi-processors: real/shadow accesses may be reordered
- Simple solution: serialise thread execution!
  - Tools can ignore the issue
  - Great for uni-processors, slow for multi-processors…

Performance

SPEC2000 Performance

Valgrind, no-instrumentation	4.3x
Pin/DynRIO, no-instrumentation	~1.5x

Memcheck	22.1x (7--58x)
Most other shadow value tools	10--180x
LIFT	3.6x (*)

(*) LIFT limitations:

- No FP or SIMD programs
- No multi-threaded programs
- 32-bit x86 code on 64-bit x86 machines only

Post-performance

- Only Valgrind allows robust shadow value tools
  - All robust ones built with Valgrind or from scratch
- Perception: “Valgrind is slow”
  - Too simplistic
  - Beware apples-to-oranges comparisons
  - Different frameworks have different strengths

Future of DBI

The future

- Interesting tools!
  - Memcheck changed many C/C++ programmer's lives
  - Tools don't arise in a vacuum

- What do you want to know about program execution?
  - Think big!
  - Don't worry about being practical at first

If you remember nothing else…

Take-home messages

- Heavyweight tools are interesting
- Each DBI framework has its pros and cons
- Valgrind supports heavyweight tools well

(Extra slides)

The past: performance

- Influenced by Dynamo: dynamic binary *optimizer*
- Everyone in research focuses on performance
  - No PLDI paper ever got rejected for focusing on performance

    "The *subjective* issues are important – ease of use and robustness, but performance is the item which would be most interesting for the audience." (my italics)
- Slow tools are ok, if sufficiently useful

Shadow value requirements

- Requirements:
  - (1) Shadow all state
  - (2) Instrument operations that involve state
  - (3) Produce extra output without disturbing execution

Robustness

- Q. How many programs can Valgrind run?
  - A. A lot
- Valgrind is robust, Valgrind tools can be
- SPEC2000 is not a good stress test!
  - Reviewer: “If the authors want to claim that their tool is to be used in real projects, then they would need to evaluate their tools using the reference inputs for the SPEC CPU2K benchmarks.”