Directed Proof Generation for Machine Code*

A. Thakur[1], J. Lim[1], A. Lal[2], A. Burton[1],
E. Driscoll[1], M. Elder[1**], T. Andersen[1], and T. Reps[1,3***]

[1] University of Wisconsin; Madison, WI, USA
[2] Microsoft Research India; Bangalore, India
[3] GrammaTech, Inc.; Ithaca, NY, USA

1 Experiments

Our experiments (see Fig. 1) were run on a single core of a single-processor quad-core 3.0 GHz Xeon computer running Windows XP, configured so that a user process has 4 GB of memory. They were designed to test various aspects of a DPG algorithm and to handle various intricacies that arise in machine code (some of which are not visible in source code). We compiled the programs with Visual Studio 8.0, and ran MCVETO on the resulting object files (without using symbol-table information).[4]

The examples `ex5`, `ex6`, and `ex8` are from the NECLA Static Analysis Benchmarks. The examples `barber`, `berkeley`, `cars`, `efm` are multi-procedure versions of the larger examples on which SYNERGY [1] was tested. (SYNERGY was tested using single-procedure versions only.) `Instraliasing` illustrates the ability to handle instruction aliasing. (The instruction count for this example was obtained via static disassembly, and hence is only approximate.) `Smc1` illustrates the ability of MCVETO to handle self-modifying code. `Underflow` is taken from a DHS tutorial on security vulnerabilities. It illustrates a `strncpy` vulnerability.

The examples are small, but challenging. They demonstrate MCVETO's ability to reason automatically about low-level details of machine code using a sequence of sound abstractions. The question of whether the cost of soundness is inherent, or whether there is some way that the well-behavedness of (most) code could be exploited to make the analysis scale better is left for future research.

References

1. B. Gulavani, T. Henzinger, Y. Kannan, A. Nori, and S. Rajamani. SYNERGY: A new algorithm for property checking. In *FSE*, 2006.

* Supported, in part, by NSF under grants CCF-{0540955, 0810053, 0904371}, by ONR under grants N00014-{09-1-0510, 09-1-0776}, by ARL under grant W911NF-09-1-0413, and by AFRL under grant FA9550-09-1-0279.
** Supported by an NSF Graduate Fellowship.
*** T. Reps has an ownership interest in GrammaTech, Inc., which has licensed elements of the technology reported in this publication.
[4] The examples are available at `www.cs.wisc.edu/wpis/examples/McVeto`.

Program		McVeto performance (x86)				
Name	Outcome	#Instrs	CE	SE	Ref	time
blast2/blast2	timeout	98	**	**	**	**
fib/fib–REACH-0	timeout	49	**	**	**	**
fib/fib–REACH-1	counterex.	49	1	0	0	0.125
slam1/slam1	proof	84	15	129	307	203
smc1/smc1–REACH-0*	proof	21	1	60	188	959
smc1/smc1–REACH-1*	counterex.	21	1	0	0	0.016
ex5/ex	counterex.	48	2	10	38	3.05
doubleloopdep/count–COUNT-6	counterex.	31	7	11	13	11.5
doubleloopdep/count–COUNT-7	counterex.	31	7	11	13	11.6
doubleloopdep/count–COUNT-8	counterex.	31	7	11	13	11.6
doubleloopdep/count–COUNT-9	counterex.	31	7	11	13	11.7
inter.synergy/barber	timeout	253	**	**	**	**
inter.synergy/berkeley	counterex.	104	5	13	16	3.95
inter.synergy/cars	proof	196	11	118	349	188
inter.synergy/efm	timeout	188	**	**	**	**
share/share–CASE-0	proof	50	3	24	75	8.5
cert/underflow	counterex.	120	2	6	12	9.55
instraliasing/instraliasing–REACH-0	proof	46	2	36	126	15.0
instraliasing/instraliasing–REACH-1	counterex.	46	2	17	55	5.86
longjmp/jmp	AE viol.	74	1	0	0	0.015
overview0/overview	proof	49	2	31	91	54.9
small_static_bench/ex5	proof	33	3	7	13	2.27
small_static_bench/ex6	proof	30	1	55	146	153
small_static_bench/ex8	proof	89	4	17	46	6.31
verisec-gxine/simp_bad	counterex.	1067	1	0	0	0.094
verisec-gxine/simp_ok	proof	1068	**	**	**	**
clobber_ret_addr/clobber–CASE-4	AE viol.	43	4	9	18	2.13
clobber_ret_addr/clobber–CASE-8	AE viol.	35	2	2	5	0.625
clobber_ret_addr/clobber–CASE-9	proof	35	1	5	21	1.44

Fig. 1. McVeto experiments. The columns show whether McVeto returned a proof, counterexample, or an AE violation (Outcome); the number of instructions (#Instrs); the number of concrete executions (CE); the number of symbolic executions (SE), which also equals the number of calls to the Yices solver; the number of refinements (Ref), which also equals the number of Pre_α computations; and the total time (in seconds). *SMC test case. **Exceeded twenty-minute time limit.